TECHNICAL REPORT
NATICK/TR-03/034

AD_____

CONRO: SELF-RECONFIGURABLE ROBOTS

by
Peter Will
and
Wei-Min Shen

Information Sciences Institute
University of Southern California
Marina del Rey, CA 90292

September 2003

Final Report
June 1998 – September 2002

Approved for public release; distribution is unlimited

Prepared for
U.S. Army Soldier and Biological Chemical Command
Soldier Systems Center
Natick, Massachusetts 01760-5020

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
06-10-2003	Final Report	June 1998 - September 2002

4. TITLE AND SUBTITLE
CONRO: SELF-RECONFIGURABLE ROBOTS

5a. CONTRACT NUMBER
C-DAAN02-98-C-4032

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Peter Will and Wei-Min Shen

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Information Sciences Institute
University of Southern California
4676 Admiralty Way
Marina del Rey, CA 90292

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Sponsor: Defense Advanced Research Projects Agency (DARPA)
Microsystems Technology Office (Elana Ethridge)
3701 North Fairfax Drive
Arlington, VA 22203-1714

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)
NATICK/TR-03/034

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited

13. SUPPLEMENTARY NOTES
Monitor: US Army Soldier and Biological Chemical Command, Soldier Systems Center,
ATTN: AMSSB-RSS-MA(N) (T. Gilroy), Natick, MA 01760-5020

14. ABSTRACT
The goal of the CONRO Project was to develop a miniature reconfigurable robot that can be tasked to perform reconnaissance and search and identification tasks in urban, seashore and other field environments. CONRO was made from small and identical modules that can be programmed to alter their topological connections and form different configurations in order to respond to environmental challenges such as obstacles and unexpected situations. A total of 20 CONRO modules were built. Each CONRO module has two batteries, one micro-controller, two servos, four docking connectors, and four sets of infrared sensors and emitters for communication and docking guidance. Some modules also have micro-camera, tilt and touch sensors, and wireless communications. These modules can be configured into snakes, centipedes, caterpillars, rolling tracks, etc., and all configurations are capable of locomotion. Current top speed of the robot is 0.6-1.0 mph and is improving continuously. Novel control software systems for self-reconfigurable robots have also been developed. The CONRO robots can perform many metamorphic actions not reported previously in robotics literature. In certain configurations, CONRO can also complete reconfigurations without any human interventions. A CONRO sidewinder robot can also move over rough terrain such as pebbles or small obstacles.

15. SUBJECT TERMS

ROBOTS	RECONFIGURABILITY	DISMOUNTED SOLDIER	BATTERIES	HAZARDOUS ENVIRONMENT
SCOUT	METAMORPHOSIS	URBAN WARFARE	TERRAIN	REMOTE SENSORS
SENSORS	RECONNAISSANCE	URBAN ENVIRONMENT	RUBBLE	HOSTILE TERRITORY
SEARCH	IDENTIFICATION	FIELD ENVIRONMENT	MINIATURE	BIFURCATION(MATHEMATICS)

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Thomas Gilroy
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	SAR	89	19b. TELEPHONE NUMBER *(Include area code)* 508-233-5855

Table of Contents

List of Figures

List of Tables

Preface

The CONRO Project has a goal of providing the warfighter with a miniature reconfigurable robot that can be tasked to perform reconnaissance and search and identification tasks in urban, seashore and other field environments. CONRO is made from small and identical modules that can be programmed to alter their topological connections and form different configurations in order to respond to environmental challenges such as obstacles and unexpected situations.

Work on the project was undertaken between June 1998 and September 2002 by the Information Sciences Institute at the University of Southern California at Marina del Rey under U.S. Army contract number C-DAAN02-98-C-4032.

A warfighter is vulnerable when entering closed environments or looking around corners. An enemy may attack at any time. A method of progressing is to direct an expendable surrogate to enter the closed room or go around the corner and send back surveillance information that will indicate whether or not forward progress is an option. This project was conceived to develop such a self-mobile sensor platform that would perform as a tool for the dismounted warrior that was light enough to be carried to battle and be deployed in hazardous situations. These situations include house-to-house urban warfare or fighting in Afghan caves where the warrior could stay behind a wall or outside the cave and send the surrogate to scout out the situation to sense the presence of bad guys. The intention was that the warfighter would control the system remotely and would send it into the hostile space to report back on what was observed in its package of sensors. The sensors carried on the platform can include normal and infrared spectrum video, seismic, temperature, barometric pressure and many other modalities. The Warfighter was to command the robot using a pointer interface implemented on PALM Pilot-style Personal Digital Assistant (PDA) and sees the sensory data on the PDA screen.

These scenarios, urban warfare and caves, both contain rubble. The ability to climb over, under and squeeze through rubble is an important ability. Conventional wheeled robots are severely restricted in their ability to operate in these environments. CONRO handles such environments by automatically shifting its shape to suit the type of terrain encountered. It was designed to operate with legs as a crab or insect while on flat terrain but has the ability to transform its shape to a snake to use a slithering action to slide through piles of stones to get to its target.

During the three years of the project, USC/ISI has built twenty CONRO modules and accomplished many novel experimentations and demonstrations. Each CONRO module has two batteries, one micro-controller, two servos, four docking connectors, and four sets of infrared sensors and emitters for communication and docking guidance. Some modules also have micro-camera, tilt and touch sensors, and wireless communications. These modules can be configured into snakes, insects, centipedes, caterpillars, rolling tracks, etc., and all configurations are capable of locomotion. The top speed of the robot is currently 0.6-1.0 mph and it is improving continuously. We have also developed novel control software systems for self-reconfigurable robots, including the biologically inspired and distributed method called Digital Hormones (US Patent Pending), the role-based control, and the human-centered control using gloves and hand-held devices. The CONRO robots can perform many metamorphic actions, such as online *bifurcation, unification*, and *behavior-shifting*, that have not been reported previously in robotics literature. In certain configurations, CONRO can also complete reconfigurations without any human interventions. At the present time, a CONRO sidewinder robot can also move over rough terrain such as pebbles or small obstacles.

Summary

The CONRO project was conceived as a response to a BAA from DARPA in their Distributed Autonomous Program. The specific exciting challenge defined there was to go through a keyhole and pick the lock. The technical challenges of such a task that are not surmountable even today, inspired us to make the smallest most functional robot possible that could cross the terrain to the door, reach a narrow aperture (the lock) squeeze through and continue advancing across the terrain to the objective. Such a task requires self-reconfiguration either from sensing the environment or on command from the Warfighter using it and the issues of how to do command control and self-reconfiguration became the focus of the effort and is described in detail in the body of this report.

We designed and built a set of modules 1" square cross section and 4 " long that had docks at each end to allow connection to become a variety of functional robotic shapes. We used the best available physically and computationally small low power microprocessor or we could find to power the system and accepted its limitations in power and functionality in order to show early success in moving and reconfiguring.

We drove the system from a central machine (a set of interacting Unix processes) to fully distributed control using the Hormone system we describe in the body of this report. We commanded the system from a keyboard and by using Virtual Reality gloves and a simple set of hand-motion commands but always with an eye towards as autonomous a behavior as possible in order to relieve the Warfighter from the task of the detailed running of the robot when he/she needs to keep hidden to avoid being shot at.

The work progressed from being able to move at snail pace with one configuration to the current work on increasing the speed beyond 1 mph in the multi-legged insect (or spider) mode, beyond 0.5 mph in snake slithering or sidewinder modes. The system has gained a 40x increase in speed over the last two years. This has been accomplished by the substantially same prototype hardware, but used with better algorithms of all kinds.

We have traveled across smooth and 2"-diameter pebbled terrain that meets some of the requirements of operating in a desert. We built and installed sensors of several kinds as a start towards an instrumented package that would relay information back to its Warfighter controller. We built, for instance, a small camera using a state-of-the-art CMOS chip for image feedback, we used sensors from Virtual Reality gloves as cats whiskers to sense impending collisions with the environment and even upside-down sensors for use in righting itself in clambering over rocks. We demonstrated autonomous reconfiguration, the first, we believe, in the literature on robots.

The work so far has successfully shown the feasibility of the many concepts needed for a research prototype of a self-reconfigurable robot. Now the task is to build an engineered prototype for deployment under the control of a Warfighter in the dust and mud of a real field test.

CONRO: SELF-RECONFIGURABLE ROBOTS

1 Introduction

The CONRO self-reconfigurable robots are built in the context of self-reconfigurable robots. Self-reconfiguration in both physical and tactical sense is a new and critical capability for many future military and civilian applications. Tactically, self-reconfiguration will allow Unmanned Air Vehicles (UAV) or Unmanned Ground Vehicles (UGV) to restructure and self-repair their organization in unexpected situations, and will enable information agents to adjust their relationship to gather timely information. Physically, self-reconfiguration will allow robots to change shapes and sizes for difficult missions, enable smart materials to morph into different structures for best performance, and facilitate the self-assembly in space and support missions such as DARPA's Orbital Express and AFOSR's micro satellites. In naval applications, submarines and aircraft carriers may be equipped with self-reconfigurable robots for inspection and repair, and they can maneuver in tight spaces that are hard to reach by humans or conventional robots. In a marine reconnaissance mission, an underwater self-reconfigurable robot may become an eel for swimming in open water, change into an octopus for grasping rocks when enter a high-energy surf zone near coast, and then spread it out as a set of small and hard-to-detect moving units once on shore. In Army's Future Combat Systems, self-reconfigurable robots can form massive and relentless robot swarms in large-scale operations, or small scout forces with shape-alteration ability to inspect dangerous caves in enemy territory.

The CONRO metamorphic robot is made from a homogeneous set of autonomous *reconfigurable modules* that can change their physical connections and configurations under computer or human command to meet the demands of the environment. Each CONRO reconfigurable module has a size of 1.0 inch^2 cross-section and 4.0 inch long and is equipped with a micro-controller, two servo motors, two batteries, four connectors for joining with other modules, and four pairs of infrared emitter/sensor for communication with neighboring modules and docking guidance. These modules are autonomous and self-sufficient and they have the docking capability and can connect and disconnect with each other and form various shapes. Figure 1 shows a number of CONRO configurations and self-reconfigurable robots, including single module in a human hand, two 9-module six-legged insects, an 8-module rolling track, and an 8-module self-reconfigurable robot in moving as a serpent (lateral undulation), a caterpillar (concertina), a sidewinder, and a coiled snake.

Ultimately, a CONRO self-reconfigurable robot can slither down between stones to locate a person or some artifact, and then smoothly morph to become a "crab" and climb over rubble. It can become a ball to roll down a hill, or a leg can be transformed into a gripper to perform a grasping operation. Using novel control approaches, such as digital hormone model [2,3,10,12,15] and role-based control [5-7], the CONRO robots can at the present time perform online *bifurcation, unification*, and *behavior-shifting* that are unique for reconfigurable systems. There is no fixed "brain" module in the CONRO robot, and every module behaves properly according to its relative position in the current configuration. For example, a moving CONRO self-reconfigurable robot can be bifurcated into pieces, yet each individual piece will "elect" a new head and continue to behave as an independent snake. Multiple snakes can be concatenated (for unification) while they are running and become a single and coherent snake. For online behavior shifting, a tail/spine module in a snake can be disconnected and reconnected to the side of the body, while the system is running, and its behavior will automatically change to a leg (the reverse process is also true). For fault tolerance, if a multiple legged robot loses some legs, the robot can still walk on the remaining legs without changing the control program. The CONRO robot can also perform self-reconfiguration in certain configurations. Figure 2 shows the steps of reconfiguration from a snake shape to a two-legged creature that can do a locomotion gait similar to the two-arm butterfly stroke in swimming. This is probably the first time any robot was shown to perform such a metamorphic action using a totally distributed control method.

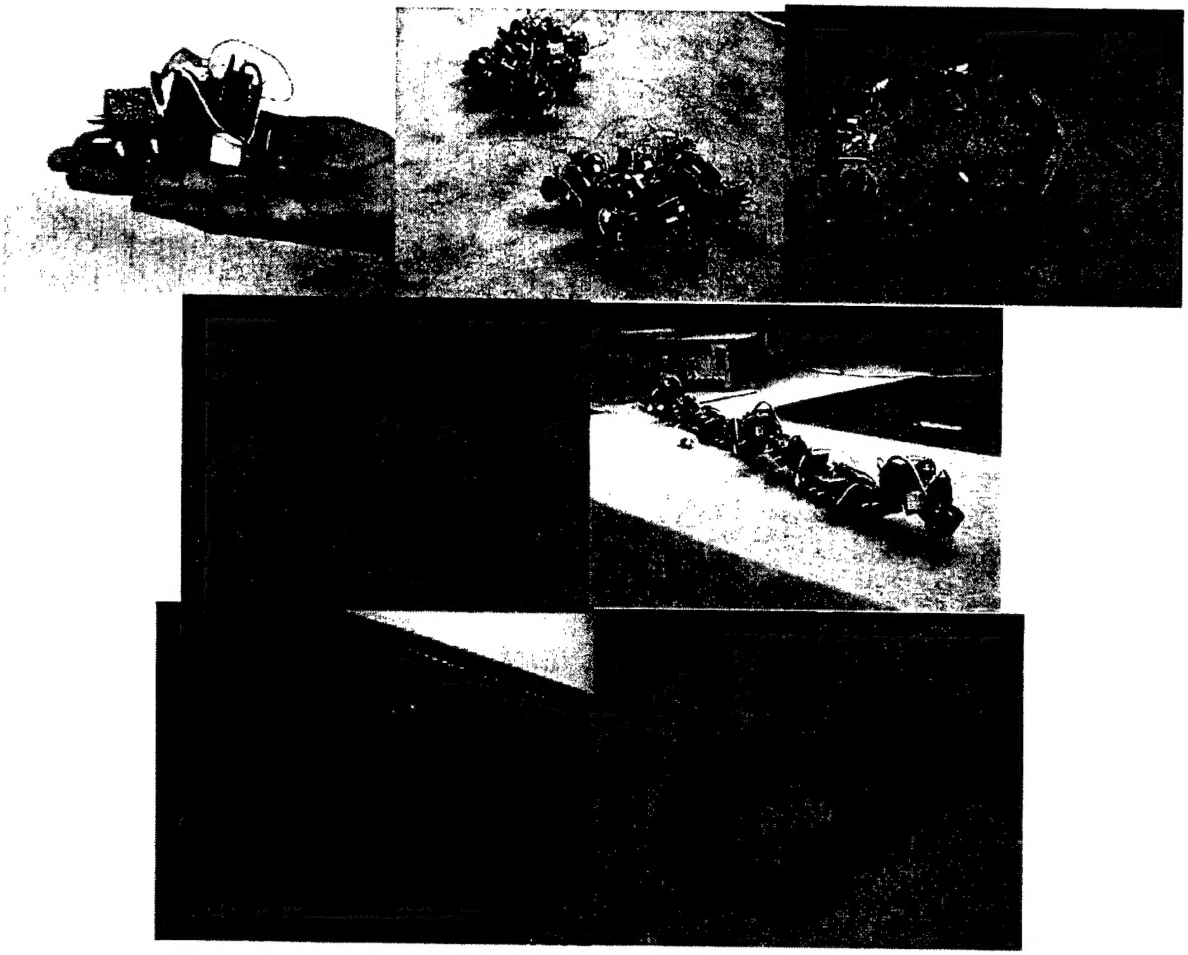

Figure 1. CONRO modules and configurations of insects, rolling track, and snakes.

Figure 2. A CONRO Robot self-reconfigures into a Butterfly-Stroke Walker.

It is a great challenge to construct and control a self-reconfigurable system such as CONRO. Each module in the robot is an autonomous and intelligent agent itself, and has power, controller, sensors, motors, connectors, and communication devices. These modules must cooperate their actions to generate the desired global effects for a given physical configuration. The concept of configuration can be interpreted in several ways. The physical interpretation is that it represents the structure or shape of the system. The connectivity interpretation is that it is a communication network topology. The control implication is that global actions (such as locomotion for a robot) require a re-computation of the local actions to be executed by the individual modules. These local actions depend on the position of the agent in the current configuration. We have focused on two general problems for

self-reconfigurable systems: (1) how agents communicate with each other where connections and configurations may change dynamically and unexpectedly, and (2) how agents collaborate local actions in the physically and tactically coupled organization. These two problems occur in many types of self-reconfigurable systems, and are critical to build a multifunctional self-reconfigurable robot. Note that ultimately the cooperative control in a multifunctional robot must be *dynamic*, to deal with the changes in organizational configuration and topology; *asynchronous*, to compensate for the lack of synchronized global clock shared by all components; *scalable*, to support ever-growing structures and shape-alteration; *collaborative*, to enable global efforts by local actions in a physically and tactically coupled organization; *reliable*, to recover from local damages and provide fault-tolerance; and finally, *self-adaptive*, to select and form the best configuration for the task and environment in hand.

The control in CONRO robots is based on a bio-inspired distributed method we call the Digital Hormone Models (DHM) (US Patent pending). The basic idea of DHM is that components in a robot form a dynamically networked organization and use hormone-like messages to communicate and collaborate global behaviors such as locomotion, tactic formation, self-reconfiguration, and self-repair. The hormone-like messages are similar, but not identical, to content-based messages. They do not have addresses but propagate through the network (note that hormone propagation is different from message broadcasting). There is no guarantee that every component in the network will receive the same copy of the original message because a hormone may be modified during its propagation. Hormones are also similar, but not identical, to pheromones because hormones can propagate from cell to cell without leaving residues in the environment. This provides a more convenient way to represent topological constraints between elements of an organization. In DHM, all components run the same control program and react to hormones based on where they are in the current configuration. For example, when receiving the same hormone, a leg module may retract, while a torso module may bend. Thus, a single hormone may propagate in the network and cause different body parts to perform different actions. This DHM has been successfully applied to the CONRO robots, including the snake configuration, and many other legged configurations. It is proved to be distributed, dynamic, robust, and efficient (running on the STAMP controller which has only 2K memory and 64 Bytes for variables). For more information about the digital hormones, please see our website http://www.isi.edu/conro.

2 Hardware Description

The goal of the CONRO project is to build deployable modular robots that can reconfigure into different shapes such as snakes or hexapods. Each CONRO module is, itself, a robot and hence a CONRO robot is actually a multi-robot system. In this section we present an overview of the CONRO modules, the design approach, an overview of the mechanical and electrical systems and a discussion on size vs. power requirement of the module. Each module is self-contained; it has its own processor, power supply, communication system, sensors and actuators. The modules, although self-contained, were designed to work in groups, as part of a large modular robot. We conclude the section by describing some of the robots that we have built using the CONRO modules and describing the miniature custom-made CONRO camera as an example of the type of sensors that can be carried as payload by these robots.

2.1 Background

Reconfigurable robots are modular robots that can change their shape. These robots could be used in applications that would benefit from the use of different or multiple fixed-size fixed-shape robots. A reconfigurable robot could change its shape into a snake to reach into narrow places during a rescue operation, into a hexapod to carry a load or it may split into many smaller robots to perform a task in parallel.

Reconfigurable robots are classified as homogeneous or heterogeneous depending on whether their modules are identical or not. In a homogeneous robot, the position of the module in the robot defines its function, e.g., the module could play the role of head, leg or spine depending on its location in the robot. In a heterogeneous robot, the function of the module defines its position in the robot, e.g., the possible positions of a leg module are restricted to the legs of the robot.

Reconfigurable robots can also be categorized according to whether or not their modules are organized in a lattice (either in the plane or 3D space). Lattice-based robots are usually homogeneous and need to reconfigure in order to move, i.e., as their topology changes, their center of mass translates accordingly. In contrast, non-lattice robots can either translate while reconfiguring or can separate their reconfiguration and locomotion stages. This separation allows them to reconfigure and then select an efficient configuration-dependent gait.

The original reconfigurable robots were designed to add versatility to the robotic manipulator. Early work by Will and Grossman, Schmitz et al., and Fukuda and Kawauchi continues to evolve in the work of Paredis and Khosla and others. Among the planar lattice-based cellular robots we find the robots based on square and hexagonal modules of Yoshida et al., and Murata et al., respectively, and the robot based on hexagons of links of Chirikjian et al. This work has been extended to 3D space, among others, by the cubic units of Murata et al., the robotic molecule of Kotay et al., the crystal module of Rus and Vona and the "I-Cube" modules by Unsal et al.. Most non-lattice based reconfigurable robots are heterogeneous. Among these we find the robots for the entertainment industry of Fujita et al., and those for space exploration of Farritor et al..

In this section, we describe the design approach and the mechanical and electrical aspects of the modules of the CONRO robots, non-lattice homogeneous reconfigurable robots, targeted to search and rescue and surveillance operations. The CONRO robots have some similarities with the Tetrobot of Hamlin and Sanderson. Both are homogeneous and can separate their locomotion and reconfiguration stages. However, the Tetrobot must be tethered while the CONRO robot is self-contained. CONRO robots are also similar to the Polypod of Yim in both capabilities and concept. However, CONRO robots emphasize size and autonomy as design parameters and are designed to support inter-robot reconfiguration, i.e., reconfiguration that involves more than one robot and leads to the merging of robots into a larger one or the splitting of a large robot into smaller ones.

This section is organized as follows. In subsection 3.2, we summarize the philosophy behind the design of the module. In subsections 3.3 and 3.4 we describe the design of the module from the mechanical and electrical points of view, respectively. In subsection 3.5 we discuss the considerations of the module with respect to its size. In subsection 3.6 we describe the resulting module and give some examples of the possible CONRO robot configurations. In subsection 3.7 we present the miniature custom-made CONRO camera built which is an example of the sensors that a CONRO robot could carry as load.

2.2 Philosophy of Hardware Design

The goal of the CONRO project is to build deployable reconfigurable robots that exhibit inter-robot reconfiguration capabilities. The capabilities of these robots are determined by the characteristics and functionality of their modules. The basic shape of the CONRO modules is that of three segments connected in a chain, as shown in Figure 3. Two independent axes of rotation, located at the intersections of these segments, provide the module with motion capabilities.

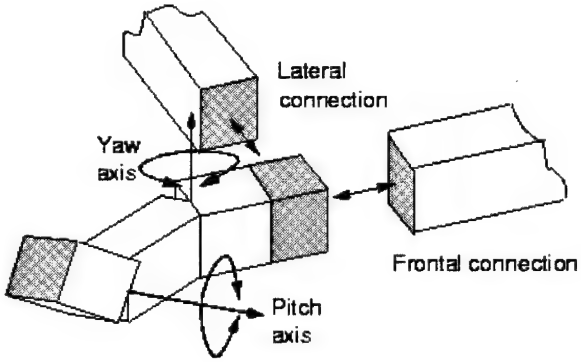

Figure 3. Basic shape of a CONRO module

The extremes of each module have ports (one on each shaded face) that allows it to connect to other modules. A detailed discussion about the philosophy of design of the CONRO module can be found in \cite{ARCas00}. The summary in this section is presented for completeness of the section.

The specifications of deployability and inter-robot reconfiguration capabilities translate into constraints on the levels of self-sufficiency, autonomy and homogeneity of the module, module size and communication capabilities. A deployable robot must be self-sufficient, i.e., capable of untethered operation. In the trivial case, an inter-robot reconfiguration split operation may create a robot formed by a single module. Therefore, to guarantee that any robot such created is self-sufficient, each module must be self-sufficient. Likewise, a module must be autonomous with respect to the use of its own resources, e.g., it has exclusive access to its sensors and actuators.

The level of homogeneity of a module determines its capabilities and the functions that it can fulfill. Each module must have a processor, power, sensors, actuators and communication systems to satisfy the self-sufficiency and autonomy constraints. Other components not needed to satisfy these constraints (e.g., cameras, antennas) can be carried by the robot as a load or are piggy-backed on a particular module, driven by a generic interface port. This trade-off between the necessary and desired components of a module reduces its design, manufacturing, testing and programming costs. All the components must fit into a package that is as small as possible to reduce the effect of inertia of the limbs and increase the relative torque-to-robot weight ratio of the actuators.

Finally, we address the communication needs of the module. During inter-robot reconfiguration, two robots need to communicate remotely to agree on the merging operation. Thus, robots need to exchange information remotely and need a mechanism to guide another robot toward itself. At the local level, each module needs to communicate with its adjacent modules. We concluded that an infrared-based system could satisfy all these requirements; it could be used for both remote and local communication and double as the directional guiding mechanism for both inter-robot and intra-robot dockings.

2.3 Mechanical Design

Our implementation of the CONRO module has three segments connected in a chain: a passive connector, a body and an active connector, as shown in Figure 4. At the intersection of the body and the two connectors there are joints that give the module yaw and pitch degrees of freedom. The weight of the module is W_m= 114g

(including batteries) and its length is L_m = 10.8*cm* excluding the length of the pins protruding from the passive connector. We now describe the parts of the module.

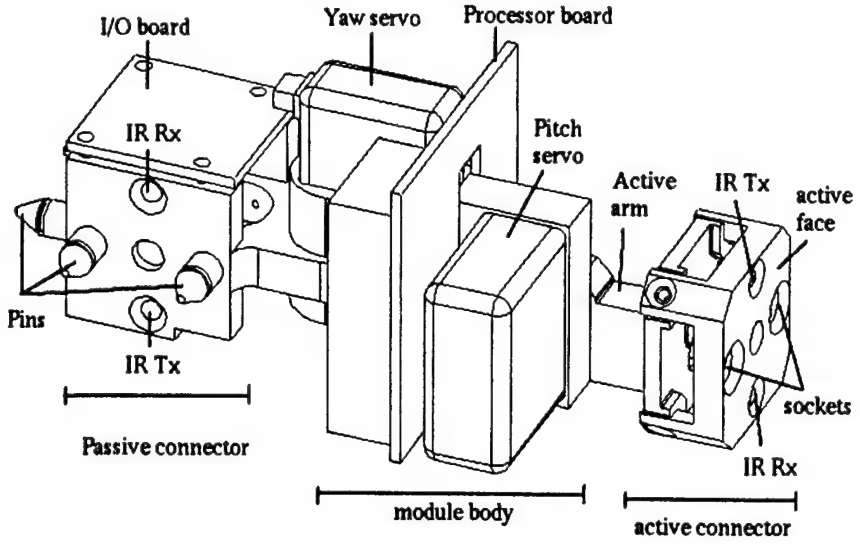

Figure 4. Parts of the CONRO module

2.3.1 The Module Body

The body is the central part of the module to which the active and passive connectors attach. It is composed of a delrin frame, two servo motors and a printed-circuit board (PCB), shown in Figure 4 as the Processor Board. The PCB has a hole in its center to allow its accommodation on the frame. When it is in place, the PCB is screwed in position to the frame. The servos fit into cavities of the frame and are held in place by friction.

Commercial-off-the-shelf servos for radio-controlled devices (Futaba S3102 RC servos) are connected directly to the processor board using 3-pin connectors. Two of the pins provide power to the servo and one carries a pulse width modulated (PWM) signal that defines the position of the shaft. Each servo has a torque of τ_a = 3.7*kg-cm*, and weighs W_a = 21*g*, mainly because of the weight of their metal gears, i.e., the servos account for 36% of the weight of the module. The output shafts of the servos are connected directly to the active and passive connectors in a direct drive fashion. The processor board is a two-layer PCB with surface-mounted components that distributes the control signals and power to the rest of the module and serves as holding place for a small 3v battery.

2.3.2 The Passive Connector

The connectors allow the module to attach to other modules. The passive connector has no moving components. Its frame is a cube of delrin with a side of 2.54 cm as shown in Figure 4. Three lateral faces of the cube have two protruding aluminum pins that fit into the sockets of the active connectors of other modules. The cylindrical pins have a lateral groove to allow the active connector to anchor to them. The particular positions of these pins and sockets permit only connections of modules that lie in the same plane, i.e., modules that are tilted 90 degrees with respect to each other cannot be connected. On each of these faces there is an infrared (IR) pair used by the module for communication and docking. The fourth lateral face of the cube has a tongue that fits on a fork of the body and allows the module to pivot the connector about the yaw axis. The yaw servo is unbiased with respect to the main axis of the module, i.e., the passive connector can rotate the same angle in the right and left directions (about 60 degrees).

The frame of the passive connector is hollow and holds the wiring of the infrared devices of the faces and the main battery of the module, a 6v battery (9g, 2.5cm height, 1.3cm diameter). The roof of the cube is a two-layer PCB that is screwed directly onto the cube. This PCB has the input-and-output (I/O) electronics that drive the infrared receivers (RX) and transmitters (TX) of the faces of the connector and doubles as the positive

contact for the battery. A 14-pin connector is used to transfer the power of the main battery to the processor board and receive the control signals for the IR components. Finally, the connector has a latch at the bottom of the cube that keeps the battery in place and serves as its negative contact. The latch can swing about one of its extremes allowing the removal of the battery. The weight of the passive connector, including the battery, is 30*g*.

2.3.3 The Active Connector

The active connector engages and disengages the pins of the passive connectors of other modules. It weighs 15g and is composed of two parts, an arm and a face, both machined in delrin, as shown in Figure 5. The body of the module is connected to the active connector by the arm. The active connector can rotate about a pitch axis located at the intersection of the arm and the body. The pitch servo is biased with respect to the main axis of the module; it can rotate 90 degrees downwards but only 30 degrees upwards. This bias allows the module to behave as the leg of a walking robot.

The face of the active connector has the same dimensions as those of the faces of the passive connector. It also has an infrared pair but the locations of the transmitter and receiver are the reverse of those of the faces of the passive connector to allow communication between modules when two modules are connected to each other.

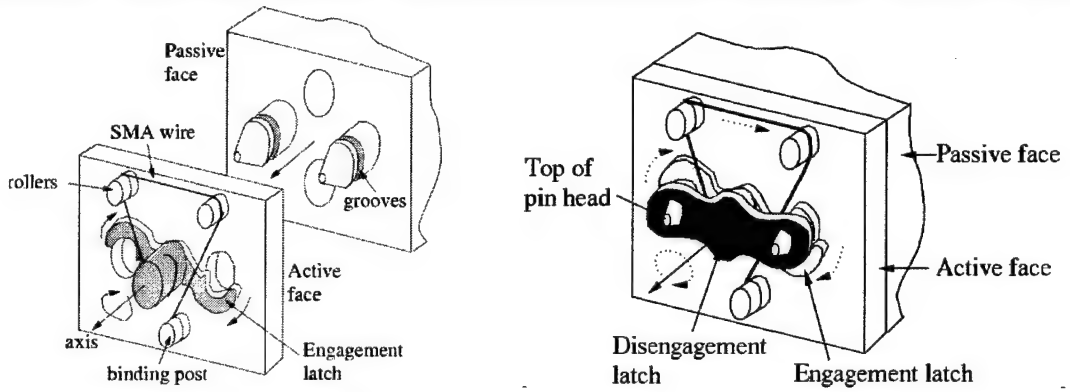

Figure 5. Stages of the docking procedure: (a) Engagement; (b) Disengagement.

The process of connecting two modules involves the active and passive connectors of the modules. Figure 5a shows a simplified view of a passive connector approaching an active connector in a docking trajectory. The active face has two sockets to receive the pins of the passive face. As the pins slide inside the sockets, their dome-shaped heads force an engagement latch to rotate in a direction perpendicular to the trajectory of the pins. Eventually, the pins are fully inserted exposing a lateral groove into which the engagement latch edge is forced by a spring action (the spring is not shown in the figure). This docking process is completely mechanical.

The process of disconnecting two modules is initiated by the active face. As shown in Figure 5a, the engagement latch can be rotated using a shape-memory alloy (SMA) wire. The wire is attached between a fixed binding post and a cylinder attached to the latch. We use two rollers to establish the path of the SMA wire and to extend its working length. When the SMA is contracted, it rotates the cylinder clockwise, against a spring, retracting the latch and freeing the pins. Using this procedure we could free the pins at any moment. However, freeing the pins is not the same as disconnecting the modules. The SMA can be activated only for a fraction of a second because it consumes a large amount of power. Thus, it is possible that the modules fail to move away from each other before the SMA is de-energized, re-engaging the pins.

The disconnection process must guarantee that the modules will be free when it is finished. In Figure 5b we show the faces of two already connected modules. We have added a view of the disengagement latch, a plate with two holes that, during engagement, allows the heads of the pins to go through. When the modules are connected, the edge of the engagement latch is pressed against the pins, into their grooves. To disconnect the modules, we contract the SMA wire as described before. As both latches rotate together, first the engagement latch frees the pins and then, the disengagement latch pushes the dome-shaped head of the pins out of the sockets. The distance that the latch pushes the pins is of the order of 0.125*mm*. Still, this displacement is enough to guarantee that the latch will not be able to re-engage the pins when the SMA relaxes. After this process, the modules are disconnected and can be moved away from each other at any moment.

2.4 Electrical Design

The electrical system of the CONRO module must support the control of the sensors and actuators, a communication system and a power system. The objectives of the design of the system are to minimize the number of discrete components, their overall weight and their power consumption while preserving the self-sufficiency and autonomy of the module.

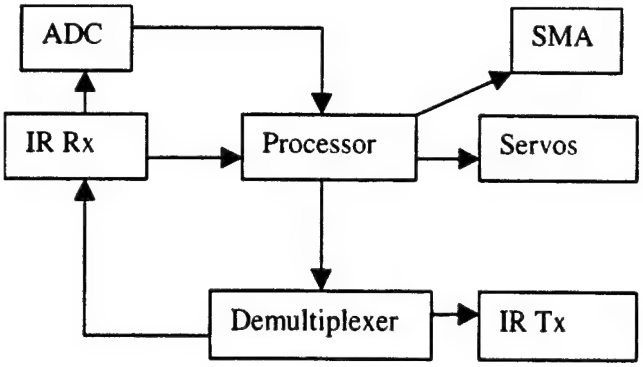

Figure 6. Functional block diagram of a module

A functional diagram of the electric system of the CONRO module is shown in Figure 6. Each module has a processor that gives it control over its sensors and actuators. The processor is defined by the use of one of three different single-chip micro-controllers: a stamp II based on a PIC16C57 processor or a stamp IIe or II-SX, both based on a SCENIX SX28AC/SS processor. The use of a zero-insertion force socket allows for the manual removal of the processor for replacement or programming. The three processors are pin-compatible but differ in speed, memory capacity, programming capabilities and power consumption. Thus, the processor of the module can be selected to suit a particular task according to its processor speed, memory and power requirements.

The processor has exclusive access to the actuators of the module. The SMA wire of the active connector is activated using a fixed current during a programmable period of time. The servos require a PWM signal generated in software because none of these micro-controllers has a dedicated PWM circuit. Hence, the processor must generate pulses every 20 ms to refresh the state of the servos.

The number of digital outputs of the processor was increased using a demultiplexer. Through the demultiplexer we can access both the IR receivers and transmitters of the module and establish serial communication with other modules. At this moment we can establish a 9600 baud link with/out flow control. The processor can route the input signal from the IR receiver to either a low-impedance input pin of the micro-controller or to a high-impedance input pin of an 8-bit analog-to-digital converter (ADC), depending on whether the infrared receiver is being used for serial communication or as an infrared sensor. This latter state is used during the docking of two modules, where a module uses its IR transmitter as a beacon and the other module uses its IR receiver as an analog directional sensor. The two modules that are docking can belong to the same robot or to two different robots. The combined use of these IR pairs provides the feedback necessary for the modules (or the robots) to approach each other and dock.

The CONRO module uses two lithium batteries: a 6v K28L battery and a 3v K58L cell, each one with a capacity of 160mA-h. The batteries set up a 9v high-voltage low-current node to power the micro-controller and a 6v low-voltage high-current node to power all other components. The use of the two batteries prevents large voltage drops at the micro-controller that would appear when components like the SMA or the servos are used. The batteries were selected for their voltage, size, weight, capacity, drain characteristics and the flatness of their discharge curves; lithium batteries are a good compromise between these features. Rechargeable batteries, although desirable for a robotics project, have an energy density that is very inferior to the lithium chemistry.

2.5 Considerations on the Module Size

Specifying the parameters of the module is difficult because of their tight coupling, e.g., battery weight, motor torque and weight, module size, operating time, etc., are all parameters that affect each other. A relationship between a set of these parameters was developed during the work preliminary to the design of the modules. We now apply these relationships to discuss the characteristics of the CONRO module.

Consider the following parameters:

V: battery voltage
C_b: battery capacity
P_m: average power consumed by the module
t: maximum operating time of the module
τ_a: actuator torque
W_a: actuator weigh
W_m: total weigh of the module
L_m: length of the module
n: number of modules that a module can lift

A simplified model of the module would relate these parameters with the following inequalities:

$$C_b \geq \frac{P_m}{V} \cdot t \tag{1}$$

$$\tau_a \geq W_m L_m \frac{(1+2n)^2}{8} \tag{2}$$

Equation 1 states that the battery must have a current delivery capacity C_b greater than or equal to that needed to supply the required average power P_m at the rated voltage V for a given period of time t. Equation 2 states that the torque of the actuator τ_a must be greater than or equal to that needed to handle n modules of weight W_m and length L_m or, equivalently, assuming that the actuator is located at the center of the module, the torque needed to overcome the inertia of (2n+1)/2 half modules, each with a weight of $W_m/2$ and length of $L_m/2$. This is the maximum torque that the actuator might need to deliver continuously.

We can use Equations 1 and 2 to estimate upper bounds of the CONRO module on n and t. The average power consumed under load by the CPU (i.e., 20 mA at 9 v), other electronics (i.e., 130mA at 6v) and each actuator (i.e., 150mA at 6v) are 180 mw-h, 780 mw-h and 900 mw-h, respectively, so the average power consumed by the module (using only one actuator) is P_m = 1860 mw-h. The equivalent battery of the module has a capacity of C_b = 160 mA-h and is rated at a voltage of V = 9v. Given an actuator torque of τ_a = 3.7 kg-cm, we find from Equation 1 that

$$t \leq \frac{C_b V}{P_m} = 0.77h$$

$$n \leq \sqrt{\frac{2\tau_a}{W_m L_m}} - \frac{1}{2} = 1.95$$

Thus, our module cannot work more than 45 mins and cannot lift more that one identical module. These estimates agree with our experience with the module, e.g., continuous operation of t is about 35 minutes and n is about one module.

2.6 The CONRO Module

We have built twenty modules that follow the description discussed in this section. Because each module is self-contained and autonomous, it is a robot in its own right and thus, it is possible to program it to execute motions and to react to stimuli. Figure 1 shows a completely untethered module in a human hand that runs a

program that rotates its connectors in sequence. Using the PIC-based micro-controller, the program can run for 35 mins continuously. Due to the high cost of running experiments using non-rechargable batteries, our robots are tested using external power supplies.

Although the modules can be run by themselves, they were designed to work in groups, connected to each other forming large robots. The priority of the modules of a robot is to communicate efficiently with their adjacent modules. The modules do not share a clock signal and thus, robot actions that require the synchronized motions of different modules rely on the quality of the communication. The programming of the communication network is complex because, due to the lack of interrupt mechanisms in our micro-controllers, the module has to poll the ports in a round-robin fashion. At this moment, the infrared communication between adjacent modules is a 9600 baud inverted serial connection with flow control. The length, format and contents of the messages depend on the specific type of control used to command the robot.

The control of a CONRO robot can be performed using a distributed control, a centralized control based on a master-slave hierarchy or a hybrid combination of both schemes. In the centralized control, the master is a remote host with a large computational capability running C/C++ code under Linux. In this particular case the messages that we are using are three bytes long and contain information about the source and destination of the message, a message identification tag and a command token along with its respective argument. There is no network description; instead, messages intended for a particular module are broadcasted and hop through the network until they reach their destination.

2.7 CONRO CMOS Digital Camera

One of the aspects of controlling these robots is providing a user interface to allow an operator to interact with the environment. Toward this goal we have developed the CONRO camera that can be mounted on the module that would work naturally as the "head" of the robot. As discussed before, the sensors carried by every module are those that we consider necessary to assure its self-sufficiency and autonomy, e.g., the IR receivers. Still, some sensors that do not need to be carried by every module are necessary. These sensors must be small and light and are either piggy-backed on a particular module or the robot carries them as a load. In this section we describe our work on producing one such sensor, the CONRO CMOS digital camera that can be carried by a module as a piggyback device. The success of this design indicates that many other relevant sensors that would fit within the tight constraints of the basic module could be produced with a focused effort, e.g., compasses, wireless links, etc.

Sensors and actuators carried by a CONRO module or robot needs to be as self-contained as possible in terms of memory, computational needs and power. Ideally, these sensors (or actuators) should work independently from the module and exchange information with the processor only when required. Likewise, if possible, they should carry their own batteries, memory and processing circuitry, i.e., they should not drain module resources. These goals are not always attainable but nonetheless, they should be taken into account in the design of these devices.

The CONRO camera, shown in Figure 7, was designed to be as self-contained as possible to minimize interaction with the resources of the module. It is small (16x16x13 mm^3), light (2.7 g), low-power (14.5 mw idle, 73 mw active) and computationally self-sufficient. Indeed, the camera, based on the VV5300 low-resolution digital CMOS image sensor chip produced by VVL, does not require external circuitry: the VVL chip is mounted on a board together with a 10 MHz clock, an EEPROM to store the camera startup configuration and the circuitry to drive the chip and sense its pixel array. A bi-directional 2-wire serial communications interface allows the device to be configured and its operating status monitored. Seven additional cables are used to interface the camera and the driving processor. Furthermore, the VVL chip has automatic gain control, which allowed us to use a single fixed-aperture lens. With our present set up, the camera provides images with a field of view of 12 degrees.

We have built color and monochrome versions of the CONRO camera. They provide an 8-bit video stream at 30 fps; we can change this rate using an on-board clock divider. Individual 164×124 raw-format images can be easily obtained from the stream. Figure 7 shows the image of a room obtained with the CONRO monochrome camera.

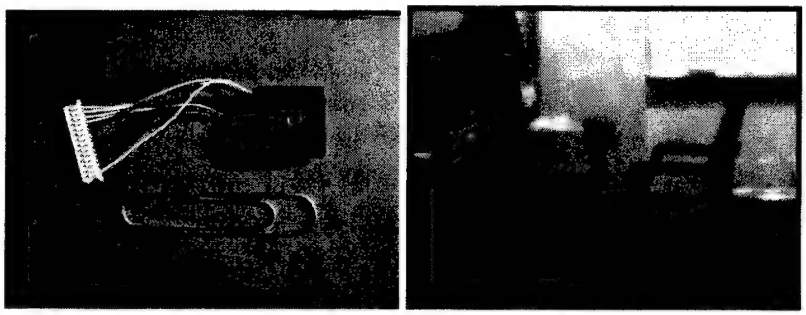

Figure 7. CONRO CMOS camera and a sample picture

Currently, the CONRO module can carry the camera but it does not have the computational resources to capture an image, (e.g., the memory of the module is 2K bytes while the image size is 22K bytes) so instead the camera is interfaced directly into the serial port of a PC. This is possible because the camera is computationally self-contained, i.e., the on-board oscillator and boot EEPROM provide the means to use the camera as a stand-alone unit, i.e., it can be used for the navigation of CONRO robots or work as a standard PC video camera.

2.8 CONRO Connectors

Each CONRO module has a set of reconfigurable connectors that can be used to dock with other modules. Such connectors should have a compliant mechanism for easy docking and a tightening mechanism for stress endurance and precision. Each connector must also have sensors to detect the status of the connector (docked or not) and to provide precision guidance in the docking process. Connectors should be able to report their statues to the onboard controller and receive commands from the controller.

The connectors must be *hermaphroditic* (genderless) so that any two connectors can dock together. To insure the correct orientation in self-reconfiguration, the connectors must be designed to allow only one possible alignment. Since space has near-zero gravity, a pulling method for docking will be much more predictable than a pushing method because any uncontrolled and non-tethered pushing force would cause the docking parties to move in an unpredictable manner.

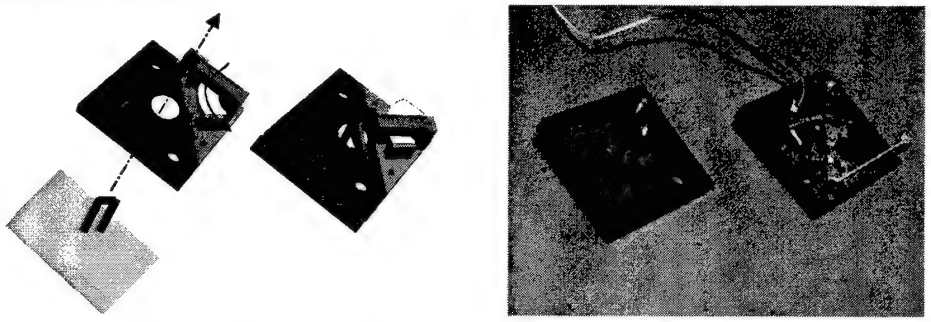

Figure 8. The CONRO-II Connector (precise and compliant)

To illustrate the concept of connectors, Figure 5 and Figure 8 illustrate two types of connectors used in the CONRO robots in the past three years. The CONRO-I type of connector (in Figure 5) consists of a male side (with two pins) and a female side (with a spring-load latch to lock the pins when they are in place). The opening of the latch is accomplished by a shape memory alloy line. The advantage of this connector is that it is precise and can sustain large stress. The disadvantage is that it has no compliant mechanism and the success of docking requires very precise movement, which is hard to obtain when the docking parties are hyper-redundant manipulator chains. The CONRO-II type connector (shown in Figure 8) is a major improvement. Inspired by the dynamic lubrication process during CONRO docking experiments, we noticed that if pins and latches are designed in a special way, then once they are "hooked", the pins can only go one-way deeper into the holes. The one-way movement needs no active driving force; the vibrations or other small disturbances in the system will be sufficient. The initial hook is high compliant and "loose," but as the pins go deeper and deeper into the hole, the

joint becomes tighter and tighter to ensure the required precision and endurance. Figure 8 illustrates one such design and implementation. On the left side of the figure, we can see that the latch mechanism behind the hole consists of a spring-loaded sloped blade that can rotate and insert into the incoming docking pin/loop. Since the blade is sloped, it can easily hook the docking loop and prevent it from moving out. After that, the blade will continue to rotate whenever there is a momentary clearance in the loop (such small momentary clearances indeed exist during vibrations or dynamic lubrication). Eventually, the blade will complete its whole rotation and the docking pin/loop will be fully rested on a flat portion of the blade and the connection will be secured and tightened (see the middle picture). To disconnect a connection, the blade will rotate in reverse driven by an especially designed *brushless* electro-magnetic motor which delivers *sufficient direct-drive torque while fitting in a small volume*, and allow the docking pin/loop to extract from the hole. This disconnecting process also relies on vibration to capture those momentary clearances when the load bearing is high. This connector has been successfully used in the CONRO robots and they make the docking process much more reliable and efficient. Note that the initial hooking can be established with a very gentle push on a triggering mechanism so the docking process of this connector is not based on pushing but the sliding force of the slopped blade entering the hook.

3 Software Control of Self-Reconfigurable Robots

This section presents a biologically inspired approach to two basic problems in modular self-reconfigurable robots: adaptive communication in self-reconfigurable and dynamic networks, and collaboration between the physically coupled modules to accomplish global effects such as locomotion and reconfiguration. Inspired by the biological concept of hormone, this section describes the Adaptive Communication (AC) protocol that enables modules continuously to discover changes in their local topology, and the Adaptive Distributed Control (ADC) protocol that allows modules to use hormone-like messages in collaborating their actions to accomplish locomotion and self-reconfiguration. These protocols are implemented and evaluated, and experiments in the CONRO self-reconfigurable robot and in a Newtonian simulation environment have shown that the protocols are robust and scaleable when configurations change dynamically and unexpectedly, and they can support online reconfiguration, module-level behavior shifting, and locomotion. The section also discusses the implication of the hormone-inspired approach for distributed multiple robots and self-reconfigurable systems in general.

Self-reconfigurable robots, in one class, are made of autonomous modules that can connect to each other to form different configurations. The connections between modules can be changed autonomously by actions of the modules themselves. Furthermore, since each module is autonomous (has its own controller, communicator, power source, sensors, actuators, and connectors), modules in a self-reconfigurable robot must collaborate and synchronize their actions in order to accomplish desired global effects. Because of this dynamism, solutions must be provided so that communication and control among modules can be adaptive to topological changes in the network.

This section addresses two basic problems for modular self-reconfigurable robots: how modules in these robots communicate with each other when connections between them may be changed dynamically and unexpectedly (thus changing their communication routing), and how these physically coupled modules collaborate their local actions to accomplish global effects such as locomotion and reconfiguration. The solutions to these problems may also be applicable to self-reconfigurable systems in general. Examples of such systems include distributed sensor networks and swarm robotic systems.

Specifically, modules in a self-reconfigurable robot must coordinate their actions to achieve given missions. Such coordination must be *dynamic*, to deal with the changes in network topology; *asynchronous*, to compensate the lack of global clocks; *scalable*, to support shape-changing and enable global efforts based on weak local actuators; and *reliable*, to recover from local damages in the system and provide fault-tolerance.

In the context of communication, a self-reconfigurable robot can be viewed as a network of nodes that can change and reconfigure their connections dynamically and autonomously. Messages in normal practice are passed between connections using named addresses (such as in the Internet) and are routed from the source to the destination. Various addressing and routing strategies are possible: Single messages can go from one module to the next one; Broadcast messages go to all nodes directly; Multicast messages go to several specific nodes. Routing may be *best-effort* as in the Internet, or *source-routed* as in some supercomputers [3]. Dynamically changing the topology requires continually determining the address and computing the route. This needs continual rediscovery of connection topology at the module level. Each module should discover and monitor unexpected local topology changes in the network, and adapt to such changes by reorganizing its relationships with other modules using their *connectors*. The concept of connector is widely applicable to many different types of networks. For example, in a supercomputing network the connectors are the channels that connect nodes to their neighbors [3]. In a wireless network, the connectors of a node are the channels available for communication. In self-reconfigurable robots, the connectors are physical so that a link is a physical coupling and a network of nodes can form physical structures with different shapes and sizes. For example, the physical connectors in CONRO must be joined and disjoined physically to change shape. Such changes in the network topology make a CONRO robot a dynamic network.

The control of the motion or locomotion of reconfigurable robotics, due to the frequent changes in topology, presents another special challenge since the action messages may need to be directed to the modules doing a specific function rather than to a specific module. Ideally, the modules should coordinate their actions by their locations in the current configuration, not by their names or identifiers. For example, the message should be sent to the "knee" module in the present configuration, not to module #37 that perhaps was the knee on the old

configuration. With this ability, a module should be able to automatically switch its behavior if its role/location is changed in the configuration. Furthermore, a control message may also require concerted actions. In other words, the message intent may be to execute an action for the robot to "go forward" rather than require the sending of several messages to swing the hip, bend the knee, bend the ankle and flex the toes and do this in spite of different modules being swapped into and out of the configuration as the system evolves.

This section presents a biologically inspired approach to address the above challenges and mimic the concept of *hormones* used among biological cells for both communication and control. A biological organism can have many hormones acting simultaneously and without interfering with each other, each hormone affecting only specific targeted sites. The main idea is that a single "hormone" signal can propagate through the entire network of modules, yet cause different modules to react differently based on their local "receptors," sensors, topology connections, and state information. Computationally speaking, a hormone signal is similar to a content-based message but has the following unique properties: (1) it has no specific destination; (2) it propagates through the network; (3) it may have a lifetime; and (4) it may trigger different actions for different receivers. Notice that hormone propagation is different from message broadcasting. A single hormone may cause multiple effects on the network and different nodes may behave differently when receiving the same hormone. Furthermore, there is no guarantee that every node in the network will receive the same copy of the original signal because a hormone signal may be modified during its propagation.

To apply this idea to adaptive communication, we view each module in a dynamic network as an active cell that can continuously discover its local topological changes and adjust its communication strategy accordingly. We design the Adaptive Communication (AC) protocol for all modules to discover and monitor their local topology and ensure the correct propagation of hormone messages in the network. This property holds regardless of the changes in the network topology.

To support distributed control with dynamic network topology, we view locomotion as the effect achieved by the interaction on the environment of executing a certain set of actions intrinsically in the robot. For instance, an automobile moves forward when the running engine is engaged with the wheels, provided among other things that there is enough friction between the tires and the road. In our robot we execute a certain set of intrinsic motions and the interaction of these motions with the environment causes locomotion. Motion execution is thus execution of module actions in the robot connection topology plus its interaction with the environment. The hormone concept described above in the context of topology discovery applies equally well to motion execution. We have designed the Adaptive Distributed Control (ADC) protocol for this purpose and applied it to the control of CONRO-like self-reconfigurable robots.

The rest of the section is organized as follows. Section 3.1 discusses the related work. Section 3.2-3.4 presents a general method for topology discovery and the AC protocol. Section 3.5-3.9 extends the AC protocol to the ADC protocol for both distributed control and adaptive communication among self-reconfigurable modules. Section 3.10 presents the experimental results of applying the hormone-inspired control protocols to the CONRO robot. Finally, Section 3.11 discusses some fundamental questions about the hormone inspired approaches and suggests future research directions.

3.1 Related Work

The communication and control of self-reconfigurable robots is a challenging problem and the approaches for the problem can be either centralized or distributed. From the viewpoint of flexibility and reliability, the distributed approaches are generally preferred for the self-reconfigurable robots. Two recent general articles [4, 5] have provided a good survey of the field.

Related work for centralized control includes Yim et al. in which configuration-dependent gait control tables are used to specify actions for each module for each step. Chirikjian et al. study the metric properties of reconfigurable robots and Chirikjian and Burdick propose a mathematical model for controlling hyper-redundant robot locomotion. Kotay and Rus propose a control algorithm for controlling molecular robots. Castano et al. use a centralized approach for controlling locomotion and discovering network topology. Rus and Vona use the Melt-Grow planner for the Crystalline robot. Unsal et al. utilize a centralized planner for bipartite self-reconfigurable modules. Most recently, Yoshida et al. and Kamimura et al. demonstrate the online reconfiguration (reconfiguration while locomotion) using a centralized method.

Related work for distributed control includes Fukuda and Kawauchi's control method for CEBOT, the series of control algorithms proposed by Murata et al. for self-assembly and self-repairing robots, the hormone-based distributed control method proposed by Shen et al., and the role-based control method by Stoy et al. (see Appendixes). Most recently, several distributed methods have been proposed for lattice-based robots, including a "secent"-based approach by Bojinov et al., a goal-ordering based approach by Yim et al., a parallel planner by Vassilvitskii et al., and an automata-based approach by Butler et al..

The distributed control method proposed in this section is different from the previously proposed distributed control methods in several aspects. First, a module selects actions based on multiple sources of local information, including the local topology, the sensory inputs, the local state variables, and most importantly the received hormone messages. Second, the local topology defined in this section distinguishes the connectors of a neighboring module and treat different connectors differently. In other words, a module knows not only that its connector c_x has a neighbor, but also the name of the connector to which c_x is connected. This provides more power for topology representation. Third, the method proposed here can deal with both locomotion and reconfiguration using the same unified framework. This has been demonstrated through the ability of *distributed* online reconfiguration on a chain-based real robot. To the best of our knowledge, such a demonstration has not been done before. Fourth, the method described here has wider application scope than the Cartesian lattice, and can support modules that have internal deforming actions such as pitch, yaw, and roll.

The concept of hormone has previously inspired several researchers to build equivalent computational systems. Autonomous Decentralized Systems (ADS) is perhaps the earliest attempt to use hormone-inspired methodology to build systems that are robust, flexible, and capable of doing on-line repair. In ADS, the Content Code Communication Protocol was developed for autonomous systems to communicate not by "addresses" but by the content of messages. The ADS technology has been applied in a number of industrial problems, and has the properties of on-line expansion, on-line maintenance, and fault-tolerance. However, ADS systems have yet been applied to self-reconfiguration. Another similar approach is proposed in where markers are passed in a network to dynamically form sets of nodes for performing parallel operations. Finally, biologically inspired control methods have also been used for robot navigation.

3.2 Adaptive Communication

As described above, the modules in a self-reconfigurable robot are reconfigured structurally. The physical interpretation of this action is that shape morphing occurs. The connectivity interpretation is that the modules have a new communication network topology. The control implication is that global actions such as locomotion require a re-computation of the local actions to be executed by the individual modules. These local actions depend on the position of the module in the reconfigured structure. To the best of our knowledge, such control approach can support some unique and new capabilities, such as distributed and online *bifurcation, unification,* and *behavior-shifting*, which have not been seen before in robotics literature. For example, a moving self-reconfigurable robot with many modules may be bifurcated into pieces, yet each individual piece can continue to behave as an independent snake. Multiple snakes can be concatenated (for unification) while they are running and become a single but longer snake. For behavior-shifting, a tail/spine module in a snake can be disconnected and reconnected to the side of the body, and its behavior will automatically change to a leg (the reverse process is also true). In fault tolerance, if a multiple legged robot loses some legs, the robot can still walk on the remaining legs without changing the control program. All these abilities would not be possible if modules could not cope with the topological changes in the communication network.

In this section, we describe an adaptive communication protocol for dynamic networks such as those used in self-reconfigurable robots. Using this protocol, modules can communicate even if the topology of the network is changing dynamically and unexpectedly. Communication with this protocol will be shown to be robust, flexible, and will allow reconfiguration while the network is in operation. The reconfiguration can either be self-initiated, superimposed by external agents, or in response to sensor interaction with the environment.

3.3 Self-Reconfigurable Modules and Networks

To illustrate the concept of adaptive communication in a self-reconfigurable network, we will use the CONRO robot as an example. A CONRO robot consists of a set of modular modules that can connect/disconnect to each other to form different robot configuration. The detail of a single module is shown in Figure 9. Each CONRO module is a generalized-cylinder that is 4.0 inch long and 1.0 inch^2 in diameter. Every module is autonomous, self-sufficient, and equipped with a micro-controller, two motors, two batteries, four connectors for joining with other modules, and four pairs of infrared emitter/sensor for communication and docking guidance.

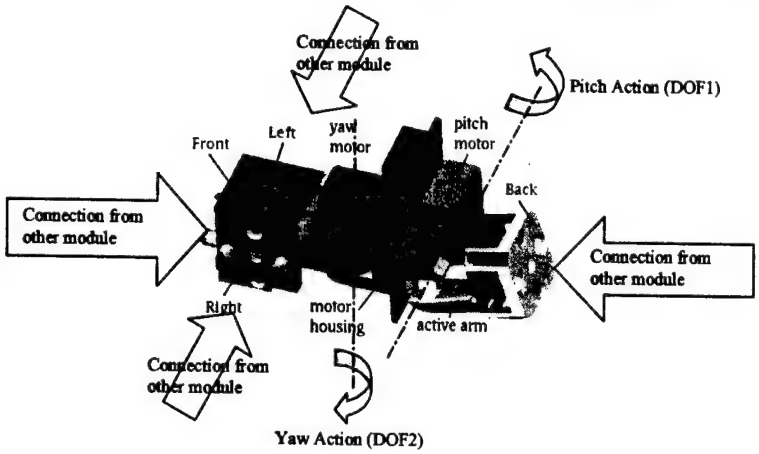

Figure 9. The schema for a CONRO module and its 4 connectors

The movements of modules are actuated by two servomotors, which provide the pitch (up and down) rotation called DOF1 and the yaw (left and right) rotation called DOF2. With these two degrees of freedom, a single module can wiggle its body and has a limited ability to move. However, when two or more modules connect to form a structure, they can accomplish many different types of locomotion. For example, a chain of modules can mimic a snake or a caterpillar, a body with legs can perform insect or centipede gaits, and a loop can move as a rolling track. Karl Sims [38] has studied this question in details and developed a system for discovering the motion possibilities of different block structures. To some extent, CONRO provides a physical implementation of his results.

The control program on a CONRO module is written in the BASIC language and is running on the on-board STAMP II micro-controller that has only 2K bytes of ROM for programs and 32 bytes of RAM for variables. Such a tight computational resource poses additional challenges for the control program. We believe that the simplicity and efficiency of hormone-inspired approach has contributed greatly to the successful implementation of all functions and programs on board.

CONRO modules can connect to each other by their docking connectors located at either end of each module. At one end, called the module's *back*, is a female connector, which has two holes for accepting another module's docking pins, a spring-loaded latch for locking the pins, and an Shape Memory Alloy (SMA)-triggered mechanism for releasing the pins. At the other end of a module, three male connectors are located on three sides of the module, called *front, left,* and *right.* Each male connector consists of two pins. When a male connector and a female connector are joined together, we call the connection an *active link.* The connected modules are called *neighbors.*

CONRO modules communicate with each other through active links. Each connector has an infrared transmitter and an infrared receiver, and they are arranged in such a way that when two connectors are joined to form an active link, the transmitter and the receiver of one side are aligned with the receiver and the transmitter on the other side, forming a bi-directional local communication link. In CONRO modules, such communication mechanism is established by a handshake between the sender and the receiver. When the sender wants to send a message, it turns on its infrared transmitter and waits for the receiver to respond. When the receiver detects the signal, it will turn on its transmitter and inform the sender and both parties will immediately enter the serial

communication protocol (RS232 with 9600 baud rate) and the message will be sent and received. If there is no receiver at the other end, then the sender will not get any response and the procedure will return a timeout failure.

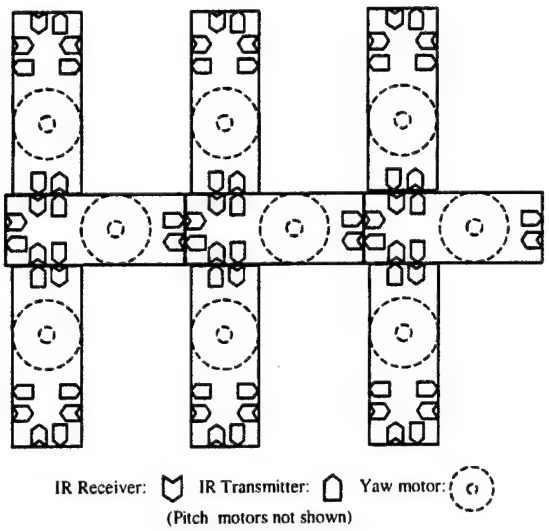

Figure 10. A top-down view of a self-reconfigurable communication network among 9 modules

Figure 10 shows a network of 9 modules (9x4 connectors) forming a hexapod. There are 8 active links (which use 16 connectors) and the rest of 20 connectors are still open. Each active link uses two pairs of aligned infrared transmitters and receivers for communication. As we can see from this example, a CONRO robot can be viewed as a communication network of connected modules as well as a physically connected set of modules.

Based on the above description, we define a self-reconfigurable communication network as a connected, undirected graph that has the following properties:

1. Each node is a self-reconfigurable module;
2. Each node has finite, named connectors. Two connectors of two modules can join and form an active link but one connector can only be in at most one active link.
3. Each edge is an active link;
4. The topology of the network may change dynamically, and active links may appear or disappear dynamically;
5. Nodes can only communicate through active links;
6. Nodes do not know the network size nor have unique IDs.

3.4 The Representation of Local Topology

We represent the local topology of a CONRO module in a self-reconfigurable network based on how its connectors are connected to the connectors of its neighbor modules. Shown in Figure 11, a module is type T0 if it does not connect to any other modules. A module is type T1 if its back connector, *b*, is connected to the front, *f*, of another module. A module is type T2 if its front connector is connected to the back of another module. A module is type T16 if its back is connected to the front of a neighbor and its front is connected to the back of another neighbor. A module is type T21 if its back is connected to the front of another module, and its left, *l*, and right, *r*, are connected to the backs of other two modules respectively. Note that every active link is a pair of the connector *b* (the only female connector in a CONRO module) and one of the three male connectors, *f*, *l*, and *r*. There are 32 types of local topology as listed in Table 1 and these types are ordered by the number of active links they have. For example, type T0 has no active links; types T1 through T6 have one active link, types T7 through T18 have two active links, types T19 through T28 have three active links, and types T29 through T31 have four active links.

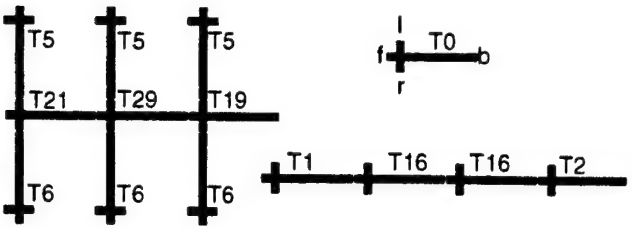

Figure 11. Examples of module topological type (T0,T1,T2,T5,T6,T16,T21,T29) (f, l, r, b are connectors)

Table 1. The Local Topology Types of CONRO Modules

CONNECTION TYPES OF CONRO MODULES										
Connected to other modules	This Module					This Module				
	b	*f*	*r*	*l*	Type	*b*	*f*	*r*	*l*	Type
					T0	*f*	*b*			T16
	f				T1	*f*		*b*		T17
		b			T2	*f*			*b*	T18
			b		T3		*b*	*b*	*b*	T19
				b	T4	*f*	*b*	*b*		T20
	l				T5	*f*		*b*	*b*	T21
	r				T6	*f*	*b*		*b*	T22
		b	*b*		T7	*l*	*b*	*b*		T23
			b	*b*	T8	*l*		*b*	*b*	T24
		b		*b*	T9	*l*	*b*		*b*	T25
	l	*b*			T10	*r*	*b*	*b*		T26
	l		*b*		T11	*r*		*b*	*b*	T27
	l			*b*	T12	*r*	*b*		*b*	T28
	r	*b*			T13	*f*	*b*	*b*	*b*	T29
	r		*b*		T14	*l*	*b*	*b*	*b*	T30
	r			*b*	T15	*r*	*b*	*b*	*b*	T31

3.5 The Adaptive Communication Protocol

Using the concept of hormone messages and local topological types defined above, we can define the Adaptive Communication (AC) protocol for continual rediscovery of network topology and ensure adaptive communication. Fig. 11 shows the pseudo-code program for the AC protocol. The main procedure is a loop of receiving and sending (propagating) "probe" hormones between neighbors, and selecting and executing local actions based on these messages. A probe is a special type of hormone that is used for continuously discovering and monitoring local topology. Other types of hormones that can trigger more actions will be introduced later. All modules in the network run the same program, and every module detects changes in its local topology (i.e., the changes in the active links) by sending probe messages to its connectors to discover if the connectors are active or not. The results of this discovery are maintained in the vector variable LINK[C], where C is the number of connectors for each module (e.g., C=4 for a CONRO module). If there is no active link on a connector *c* (or an existing active link on *c* is disconnected), then sending of a probe to *c* will fail and LINK[*c*] will be set to nil. If a new active link is just created through a connector *c*, then sending a probe to *c* will be successful and LINK[*c*] will be updated. After one exchange of probes between two neighbors, both sides will know which connector is involved in the new active link and their LINK variables will be set correctly[1].

[1] For example, if an active link is created between the connector *x* of module A and the connector *y* of module B, then LINK[*x*]=*y* for module A, and LINK[*y*]=*x* for module B. The LINK[C] variable represents the local topology type of a CONRO module. For example, a module is type T0 if LINK[f,l,r,b] = [nil,nil,nil,nil]; type T2 if LINK[f,l,r,b] = [b,nil,nil,nil]; and type T21 if LINK[f,l,r,b] = [nil,b,b,f].

The AC protocol has a number of important properties that are essential for adaptive communication in self-reconfigurable networks.

Proposition 1: Using the AC protocol, all modules can adapt to the dynamic topological changes in the self-reconfigurable network and discover their local topology in a time less than two cycles of the main loop. The updated local topology information is stored in LINK[c].

To see this proposition is true, notice that initially all LINK variables have a nil value. If a module has a neighbor on its connector c, then LINK[c] will be set properly when this module receives a probe on that connector. Since every module probes all its connectors in every cycle of the program, the LINK[c] will be updated correctly with at most two cycles.

Proposition 2: If the network is acyclic graph, then the AC protocol guarantees that every non-probe message will be propagated to every module in the network once and only once. The time for propagating a hormone to the entire network is linear to the radius of the network graph.

```
OUT: the queue of messages to be sent out;  IN: the queue of messages received in the background;
C: the number of connectors for each module;  MaxClock: the max value for the local timer;
LINK[1,...,C]: the status variables for the connectors (i.e., the local topology), and their initially values are nil;
A hormone is a message of [type, data, sc, rc], where sc is the sending connector through which the message is sent, and rc is the
receiving connector through which the message is received.

Main()
LocalTimer = 0;
Loop forever:
  For each connector c=1 to C, insert [probe,_,c,_] in OUT;
  For each received hormone [type, data, sc, rc] in IN, do:
          {  LINK[rc] = sc;
             If (type ≠ probe) then
                    SelectAndExecuteLocalActions(type, data);
                    PropagateHormone(type, data, sc, rc); }
  Send();
  LocalTimer = mod(LocalTimer+1, MaxClock);
End Loop.

SelectAndExecuteLocalActions(type, data)
{ // For now, assume that when LocalTimer=0, a module will
  // generate a test hormone to propagate to the network
  // Other possible local actions will be introduced later.
  If LocalTimer==0, then for c=1 to C, do:
          Insert [Test, 0, c, nil] into OUT;
}

PropagateHormone(type, data, sc, rc)
{ For each connector c=1 to C, do:
    If LINK[c]≠0 and c≠rc, then
          { Delete [probe, *, c, *] from OUT;
            Insert [type, data, c, nil] into OUT; // propagation
          }
}

Send()
{ For each connector c=1 to C, do:
          get the first message [type,*,c,*] from OUT,
          Send the message through the connector c;
          If send fails (i.e., time out), LINK[c] = 0.
}
```

Figure 12. The Adaptive Communication (AC) Protocol

To see that proposition 2 is true, notice that when a new message is generated (e.g., [Test,*,*,*] in Fig. 5), it will be sent to all active links from that module. When a module receives a hormone, it will send it to all active links except the link from which the hormone is received. Since the network is acyclic, the generator module can be viewed as the root of a propagation tree, where each module will receive the hormone from its parent, and will send the hormone to all its children. The propagation will terminate at the leaf nodes (modules) where there is no

active links to propagate. Since the tree includes every module, the hormone reaches every node. Since every module in the tree has only one parent, the hormone will be received only once by any module.

For networks that contain loops (cyclic graphs), the AC protocol must be extended to prevent a hormone from propagating to the same module again and again. To ensure that each hormone is received once and only once by every module, additional local information (such as local variables) must be used to "break" the loop of communication. We will illustrate the idea in the ADC protocol when we describe the control of rolling tracks, which is a cyclic network.

3.6 Hormone-inspired Distributed Control

As described above we want a distributed control protocol that is identity free but supports a module to select its actions based on its location in the network. Since hormones can trigger different actions at different site and every module continuously discoveries its local topology, such a control method can be defined based on the hormone messages.

To illustrate the idea, let us first consider an example of how hormones are used to control the locomotion of a metamorphic self-reconfigurable robot. Figure 13 illustrates a 6-module CONRO self-reconfigurable robot and its caterpillar gait. The types of modules, from the left to the right, in this robot are: T1 (the head), T16, T16, T16, T16, and T2 (the tail). To move forward, each module's pitch motor (DOF1) goes through a series of positions and the synchronized global effect of these local motions is a forward movement of the whole caterpillar (indicated by the arrow). In general, the wavelength of the gait can be flexible (e.g., a single module can craw as a caterpillar). The example in Fig. 6 shows a wavelength of four, but other wavelengths can be defined similarly.

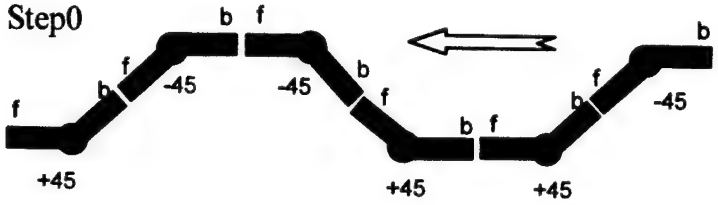

Figure 13. A caterpillar movement ('b' and 'f' are connectors, and +45 and –45 are DOF1).

To completely specify this gait, one can use a conventional gait control table [6] shown in Table 2, where each row in the table corresponds to the target DOF1 positions for all modules in the configuration during a step. Each column corresponds to the sequence of desired positions for one DOF1. The control starts out at the first step in the table, and then switches to the next step when all DOF1 have reached their target position in the current step. When the last step in the table is done, the control starts over again at step 0. As we can see in Table 2, the six columns correspond to the six module's DOF1 in Fig. 12 (the leftmost is M1, and the rightmost is M6). The first row in this table corresponds to Step 0 in Fig. 12.

Table 2. The Control Table for the Caterpillar Move

Step	Module ID for DOF1 actions					
	M1	M2	M3	M4	M5	M6
0	+45°	-45°	-45°	+45°	+45°	-45°
1	-45°	-45°	+45°	+45°	-45°	-45°
2	-45°	+45°	+45°	-45°	-45°	+45°
3	+45°	+45°	-45°	-45°	+45°	+45°

The problem of this conventional gait table method is that it is not designed to deal with the dynamic nature of robot configuration. Every time the configuration is changed, no matter how slight the modification is, the control table must be rewritten. For example, if two snakes join together to become one, a new control table must be designed from scratch. A simple concatenation of the existing tables may not be appropriate because their steps may mismatch. Furthermore, when robots are moving on rough ground, actions on each DOF cannot be determined at the outset.

To represent a locomotion gait using the hormone idea, we notice that Table 2 has a "shifting" pattern among the actions performed by the modules. The action performed by a module m at step t is the action to be performed by the module (m-1) at step (t+1). Thus, instead of maintaining the entire control table, this gait is represented and distributed at each module as a sequence of motor actions (+45°, -45°, -45°, +45°). If a module is performing this caterpillar gait, it must select and execute one of these actions in a way that is synchronized and consistent with its neighbor module. To coordinate the actions among modules, a hormone can be used to propagate through the snake and allow each module to inform its immediate neighbor what action it has selected so the neighbor can select the appropriate action and continue the hormone propagation. This example also illustrates that hormones are different from broadcasting messages because their contents are changing during the propagation.

3.7 The Adaptive and Distributed Control Protocol

To implement the hormone-inspired distributed control on the AC protocol, each module must react to the received hormones with appropriate local actions. These actions include the commands to local sensors and actuators, updates of local state variables, as well as modification of existing hormones or generation of new hormones. Modules determine their actions based on the received hormone messages, their local knowledge and information, such as neighborhood topology (module types) or the states of local sensors and actuators.

```
// Built on the AC protocol by adding a RULEBASE and extending the following procedure.

SelectAndExecuteLocalActions(type, data)
{ // Select appropriate actions based on
  // type, data, LINK, LocalTimer, and RULEBASE;
  Actions←SelectActions(type,data,LINK,LocalTimer,RULEBASE);
  For each action a in Actions, do ExecuteAction(a);
}

RULEBASE:
{ // The rules here are similar to the receptors in biological cells and are task-specific "if-then" rules as those in Table 3;
  // Although each desired task has a different set of rules, the rules can be combined together if they are not conflicting.
}
```

Figure 14. The Adaptive and Distributed Control (ADC) Protocol

Table 3. The RuleBase for the Caterpillar Move

Module Type	Local Timer	Received Hormone Data	Perform Action	Send Hormone
T1	0		DOF1=+45	[CP, A, b]
T1	(1/4)*MaxClock		DOF1=-45	[CP, B, b]
T1	(1/2)*MaxClock		DOF1=-45	[CP, C, b]
T1	(3/4)*MaxClock		DOF1=+45	[CP, D, b]
T16,T2		A	DOF1=-45	[CP, B, b]
T16,T2		B	DOF1=-45	[CP, C, b]
T16,T2		C	DOF1=+45	[CP, D, b]
T16,T2		D	DOF1=+45	[CP, A, b]

For these purpose, we specify the Adaptive and Distributed Control (ADC) protocol listed in Fig. 13. The ADC protocol is the same as the AC protocol except that there is a RULEBASE and the procedure SelectAndExecuteLocalActions() is extended to select and execute actions based on the rules in the RULEBASE. The selection process is based on (1) local topology information (such as LINK[] and the module type), (2) the local state information (such as local timer, motor and sensor states), and (3) the received hormone messages. Biologically speaking, the rules in RULEBASE are analogous to the receptors in biological cells, which determine when and how to react incoming hormones. A module can generate new hormones when triggered by the external stimuli (e.g., the environmental features such as color or sound) or by a received hormone message.

When there are multiple active hormones in the system, the modules will negotiate and settle on one hormone activity.

To illustrate the idea of action selection based on rules, let us consider how the caterpillar movement is implemented. The required rules for this global behavior are listed in Table 3. In this table, the type of the hormone message is called CP, and the data field contains the code for DOF1. The other fields of hormones are as usual, but we only show the field of sender connector (*sc*) for simplicity.

All modules in the robot have the same set of rules, but they react to hormones differently because each module has different local topology and state information. For example, the first four rules will trigger the head module (type T1) to generate and send (through the back connector *b*) four new hormones in every cycle of MaxClock, but will have no effects on other modules. The last four rules will not affect the head module, but will cause all the body modules (T16) to propagate hormones and select actions. These modules will receive hormones through the front connector *f* and propagate hormones through the back connector *b*. When a hormone reaches the tail module (T2), the propagation will stop because the tail module's back connector is not active. The speed of the caterpillar movement is determined by the value of MaxClock. The smaller the value is, the more frequent new hormones will be generated, thus faster the caterpillar moves.

Compared to the gait control table, the ADC protocol has a number of advantages. First, it supports online reconfiguration and is robust to a class of shape alterations. For example, when a snake is cut into two segments, the two disconnected modules will quickly change their types from T16 to T2, and from T16 to T1, respectively (due to the AC protocol). The new T1 module will serve as the head of the second segment, and the new T2 module will become the tail of the first segment. Both segments will continue move as caterpillar. Similarly, when two or more snakes are concatenated together, all the modules that are connected will become T16, and the new snake will have one head and one tail, and the caterpillar move will continue with the long snake. Other advantages of this hormone-inspired distributed control protocol include the scalability (the control will function regardless of how many modules are in the snake configuration) and the efficiency (the coordination between modules requires only one hormone to propagate from the head to the tail). Let n be the number of modules in the snake, then the ADC protocol requires only $O(n)$ message hops for each caterpillar step, while a centralized approach would require $O(n^2)$ message hops because n messages must be sent to n modules.

In general, the ADC protocol has the following properties:

- Distributed and Fault-Tolerant. There are no permanent "brain" modules in the system and any module can dynamically become a leader when the local topology is appropriate. Damage to single modules will not paralyze the entire system.
- Collaborative Behaviors. Modules do not require unique IDs yet can determine their behaviors based on their topology types and other local information. The global behaviors can be locomotion or self-reconfiguration.
- Asynchronous Coordination. No centralized global real time clocks are needed for module coordination, and actions can be synchronized via hormone propagation.
- Scalability. The control mechanism is robust to changes in configuration as modules can be added, deleted, or rearranged in the network.

3.8 Other Locomotion Examples of the ADC Protocol

The ADC protocol can be applied to many different robot configurations. All that is required is to provide the appropriate set of rules to the protocol and have the correct initial configuration in place. For example, Table 4 lists the set of rules that will enable a legged robot to walk. In this class of configuration, the module types are similar to those shown in Fig. 3 and Fig. 4, where a six-legged robot is shown. In other words, the left leg modules are T6, the right leg modules are T5, the head is T21, the tail T19, and the spine modules are T29. The hormone message used in Table 4 is named as LG. We use set notation such as {l,b,r} as a shorthand for the set of connectors to send the hormone. The action Straight means DOF1=DOF2=0. The action Swing means to lift a leg module, swing the module forward, and then put the module down on the ground. The action Holding means to hold a leg module on the ground while rotating the hip to compensate the swing actions of other legs.

The first two rules indicate that the head module, which can be type T21, T17, or T18, is to generate two new LG hormones with alternative data (A and B) for every cycle of MaxClock. This hormone propagates through the body modules (T29, T26, or T28) and the tail module (T19), alternates its data field, and reaches the leg modules, which will determine their actions based on their types (T5 or T6).

Table 4. The RuleBase for a Legged Walk

Module Type	Local Timer	Received Hormone Data	Perform Action	Send Hormone
T21, T17, T18	0		Straight	[LG, A, {l,r,b}]
T21, T17, T18	0.5*MaxClock		Straight	[LG, B, {l,r,b}]
T29, T19, T26, T28		A	Straight	[LG, B, {l,r,b}]
T29, T19, T26, T28		B	Straight	[LG, A, {l,r,b}]
T5		A	Swing	
T6		B	Holding	

This control mechanism is robust to changes in configurations. For example, one can dynamically add or delete legs from this robot, and the control will be intact. The speed of this gait can be controlled by the value of MaxClock, which determines the frequency of hormone generation from the head module.

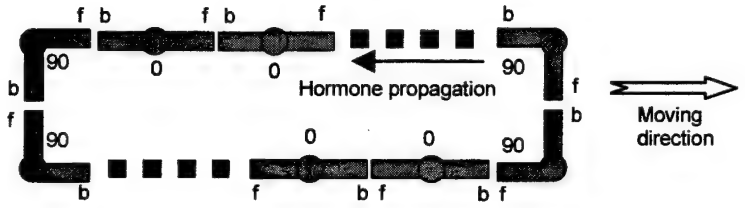

Figure 15. A rolling track configuration and movement.

As another example of how to use the ADC protocol to control locomotion of self-reconfigurable robots, Figure 15 shows the configuration of the rolling track. Notice that in this configuration, all modules are of type T16, only their DOF1 values are different. The track moves one direction by shifting the two DOF1 values (90, 90) to the opposite direction.

Table 5. The RuleBase for a Rolling Track Movement

Module Type	Local Variables	Received CP Data	Perform Action	Send Hormone
T16	Head=1, Timer=MaxClock		DOF1=90, Timer=0, Head=0	[RL,(90,90,1),b]
T16	DOF1=0	(90,90,1)	DOF1=90, Timer=0, Head=1	[RL,(90,0,0),b]*
T16	DOF1=0	(90,90,0)	DOF1=90, Timer=0, Head=0	[RL,(90,0,0),b]
T16	DOF1=0	(90,0,0)	DOF1=0, Timer=0, Head=0	[RL,(0,0,0),b]
T16	DOF1=0	(0,0,0)	DOF1=0, Timer=0, Head=0	[RL,(0,0,0),b]
T16	DOF1=90	(0,0,0)	DOF1=0, Timer=0, Head=0	[RL,(0,90,0),b]
T16	Head=0, DOF1=90	(0,90,0)	DOF1=90, Timer=0, Head=0	[RL,(90,90,0),b]
T16	Head=1, DOF1=90	(0,90,0)	DOF1=90, Timer=0, Head=0	[RL,(90,90,1),b]
Note: * means send the hormone after all local actions are completed.				

Table 5 lists the rules for a rolling track robot. The hormone used here is of type RL, and its data field contains two values of DOF1, and a binary value for selecting the head module. One hormone message continuously propagates in the loop (just as a token traveling in a token ring) and triggers the modules to bend (DOF1=90) or straighten (DOF1=0) in sequence. We assume that there is one and only one module whose local variable Head=1. This module is responsible for generating a new hormone when there is no hormone in the loop. This is implemented by the first rule, which will detect a time-out for not receiving any hormone for a long time (i.e., looping through the program for MaxClock times). The head module is not fixed but moving in the loop. We assume that the initial bending pattern of the loop is correct (i.e., as shown in Fig. 8) and the head module is initially located at the up-right corner of the loop. The rules in Table 5 will shift the bending pattern and the head position in the loop and cause the loop to roll into the opposite direction of hormone propagation. Since hormone propagation is much faster than the actual execution of actions, when a module is becoming the head, it is also responsible for making sure all actions in the loop are completed before the next round starts. The head module will hold the next hormone propagation until all its local actions (DOF1 moving from 0 to 90) are completed.

Notice that the loop configuration is a cyclic network and module types alone are no longer sufficient to determine local actions (in fact all modules in the loop have the same type T16). In general, additional local variables (such as Head) are necessary to ensure the global collaborations between modules in a cyclic network.

Due to the potential of communication errors, there may be situations where no module has the local variable Head=1 and there is a need for a new head module. In such a case, it may be possible to create a negotiation mechanism for one module to switch its local variable to Head=1, if there are none in the group -- just like some schools of fish where a female changes gender if the male in the group is dead. One possible implementation is to allow any module to self-promote to become a new head if it has not received messages for a long time. In this case, modules must negotiate among to ensure that there is one and only one head in the system. This is sometimes called the problem of *Distributed Task Selection* and we will describe a solution later.

3.9 Distributed Control of Cascade of Actions

Hormone-inspired distributed control can also be applied to the control of cascade of actions, where actions are organized in a hierarchical structure and a single action in a higher-level can trigger a sequence of lower-level actions. To illustrate the ideas, let us consider the example in Figure 16, where a CONRO robot is reconfiguring from a quadruped to a snake. The robot first connects its tail with one of the feet, and then disconnects the connected leg from the body so that the leg is "assimilated" into the tail. After this "leg-tail assimilation" action is performed four times, the result is a snake configuration. Note that the middle shape in Figure 16 is an illustration. In the real CONRO robot, at least 4 modules are needed to make a loop.

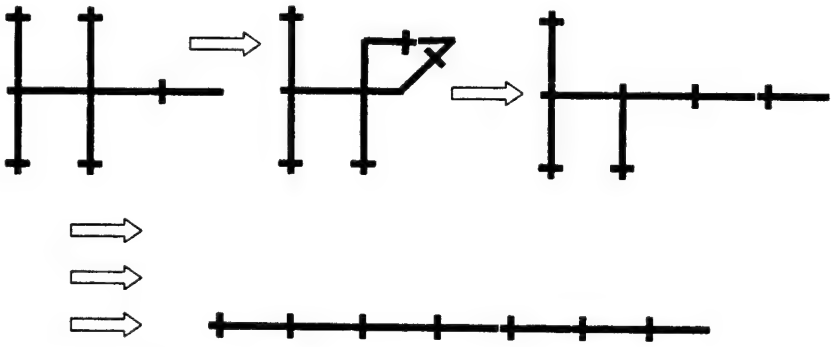

Figure 16. Reconfiguring a 4-legged robot into a snake body.

To control this reconfiguration, the high-level actions are a sequence of leg-tail assimilations, while the lower-level actions are those that enable the tail to find a foot, to align and dock with the foot, and then disconnect the leg from the body. Using hormones, the control of the reconfiguration can be accomplished as follows. One module in the robot first generates a hormone (called LTS for changing Legs To Snake). This LTS hormone is propagated to all modules, but only the foot modules (which are types T5 or T6) will react. Each foot module will start generating a new hormone RCT to Request to Connect to the Tail. Since there are four legs at this point, four RCT hormones are propagating in the system. Each RCT carries the information about its propagation path[2]. A RCT hormone will trigger the tail module (type T2) to do two things: inhibit its receptor for accepting any other RCT hormones, and acknowledge the sender (using the path information in the received RCT) with a TAR (Tail Accept Request) hormone. Upon receiving the TAR hormone, the selected foot module first terminates its generation of RCT, and then generates a new hormone ALT (Assimilate Leg into Tail) to inform all the modules in the path to perform the lower-actions of bending, aligning, and docking the tail to the foot. The details of these lower-level actions are described elsewhere [26]. When these actions are terminated, the new tail module will activate its receptor for accepting other RCT hormones, and another leg assimilation process will be performed. This procedure will be repeated until all legs are assimilated, regardless of how many

[2] A propagation path is a concatenation of all the sender connectors and receiver connectors through which the hormone has been sent so any module along the path can trace back to the original sender.

legs are to be assimilated. In Table 6, we list one possible sequence of hormone activities for assimilating four legs shown in Figure 16.

Table 6. The Hormone Activities for Cascade Actions.

Hormones	Actions
LTS	Start the reconfiguration
RCT_1, RCT_2, RCT_3, RCT_4	Legs are activated to generate RCTs
TAR, RCT_2, RCT_3, RCT_4	The tail accepts a RCT, and leg1 stops RCT_1
ALT, RCT_2, RCT_3, RCT_4	The tail and leg1 perform the assimilation process
TAR, RCT_2, RCT_4	The new tail accepts a RCT, and leg3 stops RCT_3
ALT, RCT_2, RCT_4	The tail and leg3 perform the assimilation process
TAR, RCT_2	The new tail accepts a RCT, and leg4 stops RCT_4
ALT, RCT_2	The tail and leg4 perform the assimilation process
TAR	The new tail accepts a RCT, and leg2 stops RCT_2
ALT	The tail and leg2 perform the assimilation process
Ø	No more RCT, and end the reconfiguration

3.10 Experimental Results

The hormone-inspired adaptive communication and distributed control algorithms described above have been implemented and tested in two sets of experiments. The first is to apply the algorithm to the real CONRO modules for locomotion and reconfiguration. The second is to apply the algorithm to a CONRO-like robot in a Newtonian mechanics simulation environment called Working Model 3D [39].

All modules are loaded with the same program that implements the ADC protocol illustrated in Figure 12 and Figure 14. For different configurations, we have loaded the different RULEBASE. All modules are running as autonomous systems without any off-line computational resources. For economic reasons, the power of the modules is supplied independently through cables from an off-board power supplier.

For the snake configuration, we have loaded the rules in Table 3 onto the modules and experimented with caterpillar movement with different lengths ranging from 1 module to 10 modules. With no modification of programs, all these configurations can move and snakes with more than 3 modules can move properly as caterpillar. The average speed of the caterpillar movements is approximately 30cm/minute. To test the ability of on-line reconfiguration, we have dynamically "cut" a 10-module running snake into three segments with lengths of 4, 4, and 2, respectively. All these segments adapt to the new configuration and continue to move as independent caterpillars. We also dynamically connected two or three independent running caterpillars with various lengths into a single and longer caterpillar. The new and longer caterpillar would adapt to the new configuration and continue to move in the caterpillar gait. These experiments show that the ADC protocol is robust to changes in the length of the snake configuration.

To test whether modules can automatically generate hormones when they receive appropriate environmental stimuli from their local external sensors, we have installed two tilt-sensors on one of the modules in the snake configuration, and loaded the following rules to the modules:

If tilt-sensors=[0,1], generate hormone [FlipLeft,*,*]

If tilt-sensors=[1,0], generate hormone [FlipRight,*,*]

If tilt-sensors=[1,1], generate hormone [FlipOver,*,*]

We defined the actions for FlipLeft, FlipRight, and FlipOver for all the modules so that when these hormone messages are received, the modules will perform the correct actions for DOF1 and DOF2 to flip the snake back to its normal orientation. To test this new behavior, we manually pushed the snake, while it is moving as a caterpillar, to its side or flipped it upside down. We observed that the tilt-sensors are activated, new hormones are generated, a sequence of actions is triggered, and the robot flips back to its correct orientation. (See movies at http://www.isi.edu/CONRO.)

For the legged configuration, we have loaded the rules in Table 4 onto the modules and experimented with the various configurations derived from a 6-leg robot. These configurations can walk on different number of legs without changing the program and the rules. While a 6-leg robot is walking, we dynamically removed one leg

from the robot and the robot can continue walk on the remaining legs. The removed leg can be any of the 6 legs. We then dynamically removed a pair of legs (the front, the middle, and the rear) from the robot, and observed that the robot can continue walk on the remaining 4 legs. We then systematically experimented removing 2, 3, 4, 5, and 6 legs from the robot, and observed that the robot would still walk if the remaining legs can support the body. In other cases, the robot would still attempt to walk on the remaining legs even if it has only one leg. Although we have only experimented robots with up to 6 legs, we believe in general these results can scale up to large configurations such as centipedes that have many legs.

For the rolling track configuration, we have loaded the rules in Table 5 onto the modules and experimented with rolling tracks with lengths of 8, 10, and 12. In all these configurations, the rolling track moved successfully with speed approximately 60cm/minute. The current configurations must have more than 6 modules and the number of modules must be even. This is because there must be 4 modules with DOF1=90, and at least two other modules with DOF1=0. To test the robustness of the system against loss of messages in the communication, we simulated random message losses in the program. We observed that when a message of [RL, (*,*,0), b] is lost, the robot will stop rolling momentarily and then the head module's local timer will reach MaxClock, and a new hormone will be generated and the track will resume rolling. If the lost message is [RL, (*,*,1), b], then there will be no head module in the system, and the robot will not roll again. However, since most messages are of the first kind, the chance of failing to resume rolling is low. In practice, when message losses do occur, we only observed non-recovery stops in rare occasions.

In parallel with the experiments on the real CONRO robot, we have also implemented with the ADC protocol on a simulated CONRO-like robot in a software Newtonian simulation environment called Working Model 3D [39]. Using this three-dimensional dynamics simulation program, we have designed a set of virtual CONRO modules to approximate the physical properties of the real modules, including their mass, motor torques, joints, coefficient of friction, moments of inertia, velocities, springs, and dampers. The ADC protocol is implemented in Java and runs on each simulated module. We have experimented with and demonstrated successful locomotion in various configurations, including snakes with different length (3-12 modules) and insects with different numbers (4-6) of legs. For the cascade actions, we have successfully simulated the reconfiguration sequence described in Section 3.9 using the ADC protocol.

3.11 Discussion

This section discusses some related questions about the ADC protocol: (1) How to deal with multiple hormone generators in a robot? (2) How to combine multiple rule sets and switch between them? (3) How to develop the appropriate RULEBASE for a particular global behavior? (4) Is this mechanism applicable to robotic systems in general?

In the above description, all the rules are so designed that one robot has only one hormone generator at a time. For the snake and legged robots, the generator is the head module. For the rolling track, it is the module that has a local variable head=1. When there are multiple hormone generators in a robot, modules must negotiate to select one and only hormone activity. This problem is sometimes called *Distributed Task Selection,* which is a process for modules to agree and select the same task among multiple initiated tasks in a distributed manner. To solve this problem, we have designed a distributed algorithm called DISTINCT. The main idea is to allow every activated hormone generator to compete to build a spanning tree for itself (being the root of that tree) by propagating a tree-building message to all its neighbors. During this tree building process, if a hormone generator module finds itself being asked to be a part of another tree (when it receives a tree building message from a neighbor module), it will drop its own root status and propagate that message to its neighbors (less the one from which the message is received) and become a part of that tree. If any module that is already in a tree receives another tree building message from a non-parent neighbor module, this module will select one of these received tasks, and designate itself as a new root and start building a new tree by propagating a new tree-building message to all its neighbors. This method is proved to be correct in selecting one and only one hormone activity in a distributed network.

The second question is how to combine multiple rule sets. We notice that as long as rules in both sets do not share the same conditions, then the two different rule sets can be combined into one and the switch between the two behaviors will be automatic. For example, Table 3 can be combined with Table 4 and the result rule set can

be used for demonstrating "online behavior-shifting" between caterpillar movement and leg movement. In particular, one can disconnect a tail/spine module from a snake and connect it to the side of the snake, and that module will automatically change its behavior to a leg. A similar but reverse process will change a leg module to a tail/spine module. Using this technique, we can dynamically change a snake configuration to a legged robot by rearranging modules in the body, while the robot is still running. The movies illustrated in Figure 2 provides one of such examples.

The third question is how to develop an appropriate rule set for a particular behavior. We note that the local control rules are similar to the receptors found in biological cells and they determine how modules react to hormones. At the current stage, the development of these rules requires expertise in the expected behavior and the local topological type information about the modules in the configuration. It is still an open problem how to develop these rules automatically. Approaches based genetic algorithms and other machine learning methods can be promising, but further research is needed to generate hormone receptors automatically and correctly. In general, the more complex the behavior is, the more complex the set of rules and requirements are. To make the approach feasible for obtaining for complex behaviors, a general strategy can be suggested based on Simon's hierarchical and nearly-decomposable systems. One first decomposes a complex behavior into a hierarchy of sub-behaviors and design one hormone for each of the most primitive behaviors. Then, another set of hormones is designed to compose the simple behaviors together. The hormones in Table 6 are designed using this strategy. As a direction for future research, we will develop software methods to *mechanize* these hormone design procedures.

The fourth question is whether the hormone-inspired approach described here is applicable for robotics systems in general. Although it has been the nature of self-reconfigurable robots that forced us to develop this distributed control mechanism, we believe it would be easily generalized and applied to behavior design of robots for which algorithmic, centralized approaches are usually applied (e.g. wheeled mobile robots). In particular, one can generalize the concept of "connectors between modules" to "communication channels between robots", and then the AC and the ADC protocols can be applied to controlling the collaborations among distributed robotics systems in a dynamic network. Each robot would have a number of "channels" that can be "connected" to other robots' channels to form "active links" which are not necessarily physical couplings but communication links. With this generalization, all the advantages described in this section could be beneficial to the control of distributed robotics systems. One potential concern for the scalability of the hormone-inspired protocols is that if there are delays in the communication, the system in general may behave erratically. It has been proven that in multi-agent/multi-robot systems, effects of delay may create unforeseen/emerging behavior. As a possible solution for this problem, we propose to use hormones to adjust local timers to compensate the delay. For example, one can image that when a hormone message is received, the module will readjust its local timer as a function of the hormone's lifetime (the number of hops it has been propagated). However, further experiments must be conducted to verify the effectiveness of this approach.

4 Autonomous Docking for Self-Reconfiguration

Docking is a crucial action for self-reconfigurable robots because it supports almost all practical advantages of such robots. In addition to the classic docking challenges found in other applications, such as reliable dock/latch mechanics, effective guiding systems, and intelligent control protocols, docking in self-reconfigurable robots is also subject to some unique constraints. These constraints include the kinematics constraints imposed on the docking modules by other modules in the configuration, communication limitations between the docking and relevant modules, and the demand for distributed control software because of the dynamics of configuration. To solve these challenging problems, this section reports a set of solutions developed in the CONRO reconfigurable robot project. The section presents a three-stage docking process, six different alignment protocols, distributed inverse kinematics, and other techniques such as dynamic lubrication that are essential for successful docking in CONRO-like robots. These solutions enable CONRO robots to perform autonomous and distributed reconfigurations in a laboratory environment, and they also suggest important considerations for docking in self-reconfiguration in general.

Self-reconfigurable robots are robots that can change their individual and collective shape and size in order to meet operational demands in uncertain and dynamic environments. Such robots typically consist of a collection of autonomous modules that can connect and disconnect among themselves. Since reconfigurations in these robots are achieved by changing the interconnections among modules, the actions of docking and dedocking play a crucial role for the success of self-reconfigurable robots.

A successful docking action between two connectable modules consists of at least three integrated complex stages. First, the robot must maneuver all relevant modules in the system so that the two docking modules are physically positioned close to each other. For example, for a snake-shaped robot to become a loop (connecting the head with the tail), all modules in the snake must bend their joints in order to bring the head of the snake to see the tail. Second, the two docking modules must be aligned to each other to satisfy the constraints for docking. This alignment is a complex process and must coordinate the actions of many modules in compliant with the perceived docking guidance signals. Third, once the two modules are aligned for docking, they must be pushed to establish the final mechanical/electronic connection. This final establishment typically requires movements in a precise trajectory with critical forces.

The nature of self-reconfigurable robots also poses additional complexities for the above three stages. In a self-reconfigurable robot, docking is not an action local to the two docking modules but an action global to many modules in the configuration. For example, many modules must be moved in order to bring the two docking modules together, and many modules must apply their torques in order to push the docking modules with critical forces. Docking in a self-reconfigurable robot is also a distributed action. Due to the dynamics of robot configuration, we cannot pre-determine a designated module to perform the centralized control for all possible docking situations because any damage to this module will paralyze the entire system. Finally, docking in self-reconfigurable robots must also be fast and efficient (for docking is performed very frequently in a self-reconfigurable robot), reliable (a docked connection must be as secure as a fixed connection), and energy saving (an established connection should not consume any power).

Due to these difficulties, autonomous docking remains an active research topic for self-reconfigurable robots. For example, Nilsson has designed a 2D self-aligning docking device but trading the device's generality for the tolerance of errors is still an open problem. Roufas et. al. have experimented 6D docking sensing using IR LEDs. Fukuda and Nakagawa studied docking with CEBOT. Murata et. al. constructed a complex mechanism for connecting arms. Bereton and Khosla have used visual images as guidance for docking between mobile robots. Their docking connector has a forklift and a receptacle and allows approximately 30-degree alignment errors. Their robots are skid-steered, i.e., when the forklift pins are partially in the receptacle and robot pushes straight ahead, the wheels of the robots would slip on the ground and allow the robot to center the pins in the receptacle. This feature, however, is not generally available for modular self-reconfigurable robots. Many researchers also assumed simplified solutions for docking. For example, the current prototypes of Polybots use tele-operations to assist docking. Proteo robots assume that a module can dock with another by "rolling over" onto that module and such actions are local and trivial. Robot Molecules by Rus et. al. sacrifice the low-power consumption requirement to use simple magnetic or electro-magnetic connections.

This section is a report of a set of integrated solutions to the docking problem in CONRO-like self-reconfigurable robots. Although the current experiments are for 2D docking, the challenges listed above (such as global cooperation, distributed control, speed and efficiency) are all present and the integration of all component solutions into a working system is a non-trivial matter. The section discusses solutions for alignment, trajectory following, distributed inverse kinematics, and other issues in docking. Section 4.1 describes the hardware of CONRO docking mechanism and guidance system. Section 4.2 presents the software architecture for distributed control of docking. Section 4.3 gives a brief discussion on the dedocking process. Section 4.4 presents the method for distributed inverse kinematics. Section 4.5 reports docking experimental results in CONRO and compare six different alignment strategies. Finally, Section 4.6 concludes the section with a set of future research directions.

4.1 CONRO Docking/Guidance Hardware

As we described before, CONRO modules can connect to each other by their docking connectors located at either ends of a module. At one end, three male connectors are located at the three sides of the module (they are called *Front*, *Left*, *Right*), each of which consists of two docking pins. At the other end, a female connector is located at the tip end of a module (it is called *Back*), which consists of two holes for accepting other module's docking pins. This female connector has a locking/releasing mechanism behind the holes, and can have two states. In the default or non-active state, it can accept and lock the incoming pins by a spring motion. In the activated state, it can release the lock by triggering a SMA actuator. The connector/leasing mechanism is power efficient and it consumes no electric energy when in the default state. For detailed design and implementation of these docking connectors, please see Section 2.3.

CONRO modules communicate with each other using infrared transmitters and receivers. At each connector, there is an infrared emitter/receiver pair located between the pins/holes. When a module is connected to another module through a connector, the two pairs of infrared emitter/receiver at the docked connectors will be aligned to form a bi-directional infrared communication link. Since there are four connectors for each module, there can be up to four communication links for each module.

The infrared emitter/receivers are also used as sensors for guiding two modules to align each other during a docking action. When two modules are in the range of the infrared signals, they can measure the strength of the received the signal and use the measurements to estimate the quality of the alignment (i.e., orientation and distance) between two modules. The correlation between the signal measurement and the quality of the alignment is straightforward. With fixed orientations of the two modules, the measurement corresponds linearly with the distance between the modules. Similarly, with a fixed distance between the two modules, the measurement corresponds to the orientations of the modules.

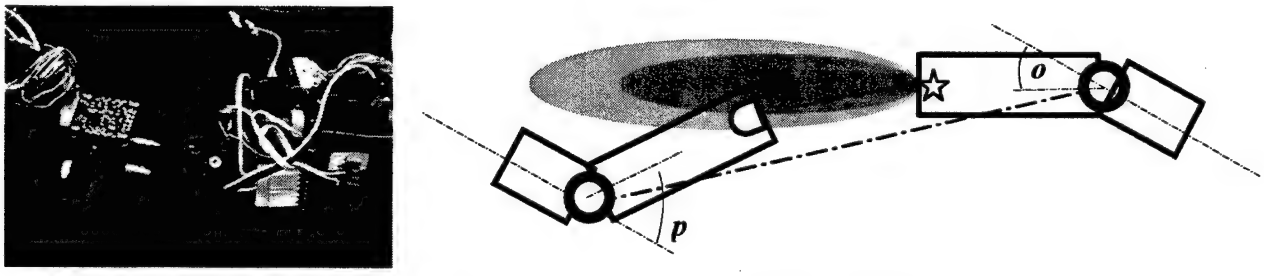

Figure 17. The measurement of docking signals

Figure 17 shows two CONRO modules in a close distance range and illustrates the docking signal in a similar situation, where o and p are the orientation angles of the two modules, d the distance between them, and the dashed line indicates the desired alignment in this particular situation. One uncertain element in this guidance system is the shape of infrared lobe from the emitter. There is no guarantee that a given emitter will produce an ideal lobe (shown in Figure 17) that points to the correct direction and that has a smooth gradient field with a single maximum peak. To make a successful docking, the lobe of the signal must be tuned and the control protocol must take this uncertainty into consideration. Furthermore, according to the features of infrared

transmitting and receiving, adjusting one docking module alone can only result in a local maximum in the alignment. In order to achieve the global optimal alignment, both docking modules must adjust their orientations collaboratively.

4.2 Docking Control for CONRO Robots

As we mentioned earlier, there are three stages in a docking process. First, two to-be-docked modules must move close enough so that they can sense each other's docking guidance signals. Second, the two docking modules must use the docking guidance signals to maneuver their locations and orientations so that their connectors are aligned and close to each other. Finally, the modules must push the connectors into each other so the mechanical mechanism can be securely latched and locked. Notice that a unique challenge in docking is that the two docking modules are not free-floating elements, but constrained as two parts of the same body. Thus, the movement of these modules may be related and can influence each other. As a result, all three stages must involve many modules in addition to the two to be docked in the configuration.

In the following discussion for the three stages, all communication between modules is subject to the constraint that modules can only talk to their immediate neighbors. Thus, a message from the head of a snake robot to the tail module must relay through all modules in the body. This communication constraint forces us to prefer a local and distributed control method to a centralized control method so that we do not need to assume any pre-designated module as a brain-like control center.

4.2.1 Maneuvering for alignment preparation

To bring two modules close enough for docking, we assume that the two docking modules are connected through a "docking chain" of modules in the current configuration, and this chain must be long enough so that it can bend to allow the two docking modules sense and touch each other. If we view a robot configuration as a graph with modules as nodes and connections as edges, then the docking chain is the shortest path in the graph between the two docking module nodes.

Let L be the number of modules in the docking chain and assume all modules in the docking chain are connected "straightly" through front-back connectors, then each module must bend approximately $2\pi/L$ to allow the two docking module to see and touch each other. In general, however, not all connections are straight, and a connection can be perpendicular either "positively" (if it branches out on the same side of the bending direction), or "negatively" (if it branches out on the opposite side of the bending direction). Since a positive connection contributes $+\pi/2$ to the overall bending, while a negative one contributes $-\pi/2$, each module in a general docking chain should bend

$$(4+m-n)\pi/2L$$

where m and n is the number of "positive" and "negative" connections in the chain, respectively. Notice that when m=n=0, then the above equation gives $2\pi/L$, which is the expected value for a straight chain.

Let B_{min} and B_{max} be the minimal and maximal bending angle for each module, then the necessary condition for bending the docking chain to allow the two docking modules to sense and touch each other is

$$B_{min} \le (4+m-n)\pi/2L \le B_{max}$$

This necessary condition for docking can be used to check if two modules in a given configuration can dock with each other or not. Given any pair of modules in a configuration graph, one can find the docking chain in the graph using the minimal spanning tree algorithm. The values of L, m, and n can then be identified based on the length of the chain and the types of connections involved in the chain. Again, all these computations can be achieved using the powerful, distributed hormone-based control framework.

4.2.2 The leader-follower alignment protocol

Once the modules in the docking chain are bent using the method described above, the two docking modules are able to sense each other's docking guidance signals. To accomplish a proper alignment, however, the two docking modules must both work actively and collaboratively. This is different from many docking tasks in classic applications, such as a spacecraft docking to a space station or a truck docking to a loading deck, where one side of the docking can be assumed static while the other can move freely and independently. In self-

reconfigurable robots, however, the movements of two docking modules are limited and constrained by the current robot configuration (all other modules are assumed to be held rigid), and an alignment cannot be achieved unless both sides of the docking adjust their orientations and positions. In a 2D environment, for example, unless both docking modules adjust their orientations (o and p), the desired alignment cannot be achieved.

To solve this problem, we have investigated a number of different search strategies that adjust the orientations of the two docking modules in some joint fashion. One approach that works well is to engage the two modules in a leader-follower relationship and adjust their orientations jointly in a hill-climbing search for the maximum measurement of guidance signals. The basic idea is that whenever the leader module (which could be chosen arbitrarily) rotates to an orientation o, it will ask the follower module to perform a scan and find/report an orientation p for the follower that gives the best guidance signal measurement m_{op}. The value p and m_{op} indicate the goodness of the leader's orientation o in the context of aligning with the follower. In the following description, we call the above procedure "find the best alignment for the follower" (FBAF) and assume the procedure FBAF(o) returns a pair of values (p, m_{op}). We also use the notation (o, (p, m_{op})), where (p, m_{op})=FBAF(o), to represent a joint alignment (o, p) and its goodness m_{op}. Using these measurements, the leader can decide to which direction it should move its orientation in order to perform an effective search for the best alignment. It can also be used to decide when the alignment process can be determined.

To decide the direction for orientation search, the leader first evaluates two consecutive joint alignments (o, (p, m_{op})) and (o', (p', m'_{op})), where $o'=o+\Delta$. The value of Δ represents the size of increment in orientation change, and the sign of Δ represents the direction of orientation change. If $m'_{op}>m_{op}$, then the leader selects Δ as its search direction. If $m'_{op}<m_{op}$, the leader selects $\Delta=-\Delta$ as its search direction. If $m'_{op}=m_{op}$, the leader repeats the above process until $m'_{op}\neq m_{op}$

Once the direction of orientation search is determined, the above procedure can be used for the leader to find the best alignment by hill-climbing. Let (o,(p,m_{op})) be the current alignment. The leader increments its rotation $o'=o+\Delta$ and obtains (p',m'_{op}) for o' from the follower. If $m'_{op}\geq m_{op}$, which indicates that o' gives a better or equal alignment than o, then the leader sets $o=o'$, $p=p'$, $m_{op}=m'_{op}$, and continues the research. If $m'_{op}<m_{op}$, which indicates a decrease in the quality of alignments, then the leader determines the search and declares (o,p,m_{op}) as the best alignment for the current distance.

Once the best alignment has been found, the two docking modules must move towards each other in the trajectory specified by the alignment. This will reduce the distance between the two modules and increase the value of m_{op}. Since the movements in self-reconfigurable modules are inevitably noisy and uncertain, a new alignment must be performed after the distance is reduced. This alignment-then-move-closer action will be repeated until the value of m_{op} is above a threshold. This indicates that the two connectors are aligned and close enough, and the final pushing stage can begin.

4.3 Establishing the Final Connection

The final pushing stage in the docking is to have the two modules to push each other in the trajectory specified by the alignment. Such movements, which are also used to reduce the distance between docking modules during alignment, require multiple modules to coordinate their actions using inverse kinematics. For example, imagine that the two docking modules in Figure 17 are aligned along the dashed line. In order to move the left (right) docking module along the alignment line, the movements of the modules behind the left (right) docking module must be coordinated using inverse kinematics. It is such coordinated movements that provide the necessary pushing force for the finally docking stage. Since the movement must not deviate from the trajectory, the step of the movement must be sufficiently small in order to ensure the smoothness of the movement. It is interesting that the computation of inverse kinematics can be distributed among modules, as we will see in the next section.

The final pushing must also overcome the friction between the docking pins and their corresponding sockets and latches. This friction is significant in self-reconfigurable robot because the strengths of modules' motors are typically limited. To overcome this problem, we have adopted the idea of dynamic lubrication. During the latching and locking stage, the two docking modules are also performing high-frequency small "shaking"

movements while pushing forward. Such movements significantly reduce the friction during docking and ensure that the docking pins to be securely locked by the spring latch behind the holes.

4.4 Distributed Inverse Kinematics

One action that is frequently used in docking is to move a docking module on a given trajectory. For example, after two docking modules are aligned, they must move along their tip directions respectively to reduce the distance between them. Similarly, such movement is also used to push the docking pins into the holes and lock them by the spring latch.

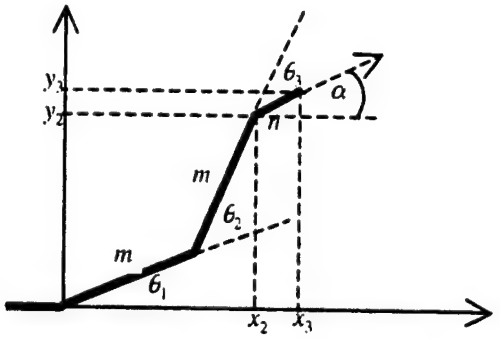

Figure 18. Inverse kinematics for three modules

To generate such actions, a chain of modules that is immediately connected to the docking module must coordinate their movement using inverse kinematics. This is similar to the classic robotics problem of controlling a robot arm to put a peg into a hole. The difference is that each module in a reconfigurable robot is an independent and autonomous system, while in classic robotics applications all motors and sensors are controlled by a single computer.

To distribute the computation for inverse kinematics among reconfigurable modules, let us consider a three-module chain illustrated in Figure 18. Given the current angles of each module θ_1, θ_2, and θ_3, and the lengths of the relevant segments m and n, we can compute the tip position (x_3, y_3) using the forward kinematics as follows:

$$x_3=m\text{Cos}(\theta_1)+m\text{Cos}(\theta_1+\theta_2)+n\text{Cos}(\theta_1+\theta_2+\theta_3) \quad (3)$$

$$y_3=m\text{Sin}(\theta_1)+m\text{Sin}(\theta_1+\theta_2)+n\text{Sin}(\theta_1+\theta_2+\theta_3) \quad (4)$$

$$\alpha=\theta_1+\theta_2+\theta_3 \quad (5)$$

Assume the tip is to be moved along the α direction by a distance δ, then the new tip position (x'_3, y'_3), the new middle position (x'_2, y'_2), and the new module angles θ'_1, θ'_2, and θ'_3, can be computed as follows:

$$x'_3= x_3+\delta\text{Cos}(\alpha) \quad (6)$$

$$y'_3= y_3+\delta\text{Sin}(\alpha) \quad (7)$$

$$x'_2= x'_3-n\text{Cos}(\alpha) \quad (8)$$

$$y'_2= y'_3-n\text{Sin}(\alpha) \quad (9)$$

$$W=\text{atan}(y'_2 / x'_2) \quad (10)$$

$$\theta'_1=W+\text{acos}[\, x'_2 / 2m\text{Cos}(W)] \quad (11)$$

$$\theta'_2=2W - 2\theta'_1 \quad (12)$$

$$\theta'_3= \alpha - (\theta'_1+\theta'_2) \quad (13)$$

Let the three modules from the origin to the tip of the chain be named as M1, M2, and M3, then under the constraints that modules can only communicate with their immediate neighbors, the above computation can be distributed among the three modules as follows:

M1 computes $A=m\text{Cos}(\theta_1)$ and $B=m\text{Sin}(\theta_1)$ and sends $[A,B,\theta_1]$ to M2;

M2 computes $\phi= \theta_1+\theta_2$, $C=A+m\text{Cos}(\phi)$ and $D=B+m\text{Sin}(\phi)$ and sends $[\phi,C,D]$ to M3;

M3 computes (3)-(9) and sends $[x'_2, y'_2]$ to M2;

M2 computes (10) and sends $[W, x'_2]$ to M1;

M1 computes θ'_1 by (11) and sends $[\theta'_1]$ to M2;

M2 computes θ'_2 by (12) and sends $[\theta'_1+\theta'_2]$ to M3;
M3 computes θ'_3 by (13).

This example illustrates that by sending intermediate results to neighbor modules, a chain of modules can compute the inverse kinematics in a distributed fashion. This style of computation is both desirable and sometimes necessary for self-reconfigurable robots because no single module can be assumed always available as the control center and every module is independent, autonomous, and has very limited on-board computation resource.

4.5 The Process of Dedocking

Compared to docking, dedocking is a relatively simple process because no alignment is needed. When a robot decides to disconnect an existing connection, it releases the latching mechanism by activating the SMA of the female connector for a short period of time. During this time, the robot "pulls" the two dedocking modules apart using the inverse kinematics method described above with a negative value for the distance parameter δ. The robot can check if a dedocking is successful or not by measuring the guidance signals between the two dedocking modules. If the measured value is below a threshold in comparison with the value when the connection is in place, then the dedocking succeeds and the robot can treat the two modules as disconnected.

4.6 Experimental Results

The above techniques and methods are implemented on CONRO reconfigurable modules, and we have conducted experiments on a 7-module snake robot to dock its head with its tail on a regular office table surface. We have made 10 runs each starting from a random initial state of being a straight snake and each run includes all three stages of docking. The average speed is approximately 3minutes/docking, and the successful ratio is 80%. The experiments are autonomous (i.e., all programs are on the modules) and modules are powered by an external power supply and triggered by a single message issued through an external command line.

Table 7. Comparison of alignment protocols

Leading schema	Rotating around the point of		
	Joints	Internal Signals	External Signals
Leader and Follower	(56±5) (7±3)	(46±10) (21±8)	(40±10) (34±9)
Alternate Leadership	(54±5) (11±3)	(43±5) (29±9)	(41±7) (38±10)

We have also conducted a set of experiments to compare the difference between six alternative alignment search protocols (see Table 7). Each experiment is a single increment in the search process although it may involve many movements of modules. All experiments are from the same initial condition and repeated for 10 runs. In Table 7, each entry reports (1) the quality of the resulting alignment (i.e., the signal value measured at the end of the experiment), and (2) the number of module movements in each experiment, which is directly in proportional to the time needed to complete the alignment. As we can see, the protocol of leader-follower with rotating around the joints of modules finds the best alignment and requires the least number of movements and time.

5 Recommendations

The work presented here represents work done on self-reconfiguration in a University research environment. The work is sound, has proven the concepts, although much more research can always be done. The main recommendation here is to say that the work should be taken to the next step of a field test where it will be tested for Field Utility by being carried to the field by a Warfighter and tasked to enter a restricted space and report on what is found. Such a test will wring out the functions needed to make the system a partner to the Warfighter.

Going to a field test requires that the system be subjected to more rigorous engineering approach. The present prototype is a research one and is not optimized by professional engineers; the prototype is open to the atmosphere, a soldier needs a system that can operate in the mud of Vietnam, the desert of Kuwait, caves in Afghan, the urban warfare environment of WWII Berlin, or the World Trade Center. The present prototype uses ordinary wires where the professional prototype would flex circuits; the present prototype is incrementally driven, the professional prototype needs to know its geometric position and have a geographic world model. The command, control and motion of the system needs to provide more power in all dimensions per cubic volume. The present system uses low cost motors from radio-controlled airplanes, a new system should use state-of-the-art miniature neodymium iron boron magnet servo-motors. The present system uses a processor with 4k ROM that cannot support an operating system and has no interrupts for real time control; the new system needs a full real-time operating system. The computer field has advanced since the present hardware design was completed and advantage needs to be taken of the advances.

Similarly for sensors, miniature GPS chips were only a gleam in the designer's eye when the present CONRO was designed but the huge gains in miniaturization of sensors for mobile telephones (now including cameras, GPS, color displays, speech recognition good enough for verbally commanding the system and many other functions) promise material gains in future miniature reconfigurable robotics.

What we learned in building the CONRO robot makes us confident in building new generations of multifunctional robots for future compact systems. There are many improvements of CONRO that are possible. The current module weighs 110 gm, including batteries, which last about 35 minutes in a continuous locomotion. A CONRO module is self-contained but has no outer protective shell. The sensor and processing hardware was circa 1998 microprocessor technology and thus had limited functionality (32 variables and 2K flash memory and limited processor power). Many currently available capabilities were not available to us, e.g. GPS and miniature cameras that have been developed for the commercial cell phone industry. The same comment applies to many other devices including sensors, especially MEMS sensors. We estimate that we can reach our design goal of less than 30 gm per module by using newer 2002 technology including: new custom motors that use Neodymium-Iron-Boron magnets, a lighter structural material than the delrin material that we used in CONRO, better chip packaging technology (no zero-insertion force sockets), thin rather than normal printed circuit boards, and omitting the apparatus and space needed for docking used in reconfiguration. We expect that these improvements will result in a much longer battery life and a ready-to-deploy system for military applications.

In summary, the research prototype shows promise of providing a Warfighter with a new tool for the challenges of the new century of terrorism. The recommendation is that we harden the system for the challenges of the future missions.

This document reports research undertaken at the U.S. Army Soldier and Biological Chemical Command, Soldier Systems Center, Natick, MA, and has been assigned No. NATICK/TR-03/034 in a series of reports approved for publication.

6 References

Arkin, R.C., Homeostatic Control for a Mobile Robot: Dynamic Replanning in Hazardous Environments. Journal of Robotic Systems, 1992. 9(2): p. 197-214.

Bererton, C., P. Khosla. Toward a Team of Robots with Repair Capabilities: A Visual Docking System. in Seventh International Symposium on Experimental Robotics. 2000.

Bonabeau, E., M. Dorigo, G. Theraulaz. Swarm Intelligence: From Natural to Artificial Systems. 1999: Oxford University Press.

Bojinov, H., A. Casal, and T. Hogg. Multiagent Control of Self-Reconfigurable Robots. in International Conference on Multi-Agent Systems. 2000.

Burdick, J., J. Radford, G.S. Chirikjian, *A Sidewinding Locomotion Gait for Hyper-Redundant Robots.* Advanced Robotics, 1996. **9**(3): p. 195-216.

Butler, Z., K. Kotay, D. Rus, K. Tomita. Generic Decentralized Control for a Class of Self-Reconfigurable Robots. in Intl. Conf. on Robotics and Automation. 2002. Washington DC.

Castano, A., W.-M. Shen and P. Will, "CONRO: Towards Deployable Robots with Inter-Robot Metamorphic Capabilities." Autonomous Robots Journal, Vol. 8, No. 3, pp. 309-324, Jul 2000.

Castano, A., and P. Will, Mechanical Design of a Module for Autonomous Reconfigurable Robots, Proc. IEEE/RSJ Intl. Conf. Intel. Robots Systems, pp. 2203-2209, Nov. 2000.

Castano, A., R. Chokkalingam and P. Will, Autonomous and Self-Sufficient CONRO Modules for Reconfigurable Robots, Proc. 5th Int'l Symp. Distributed Autonomous Robotic Systems, Springer-Verlag, pp. 155-164, Knoxville, Oct. 2000.

Castano, A., and P. Will, Representing and Discovering the Configuration of CONRO Robots, *Proc. of Intl. Conf. on Robotics and Automation*, Seoul, Korea. May. 2001.

Chirikjian, G.S., J.W. Burdick,, *The Kinematics of Hyper-Redundant Robotic Locomotion.* IEEE Trans. on Robotics and Automation, 1995. **11**(6): p. 781-793.

Chirikjian, G.S., A. Pamecha, I. Ebert-Uphoff, Evaluating efficiency of self-reconfiguration in a class of modular robots. Journal of Robotic Systems, 1996. 13(5): p. 317-338.

Chodrow, R. E., and C. R. Taylor. "Energetic cost of limbless locomotion in snakes." Federation Proc. 32:422. 1973.

Elwood, J.R.L. and D. Cundall. Morphology and behavior of the feeding apparatus in Cryptobranchus alleganiensis (Amphibia: Caudata). *Journal of Morphology* 220: 47-70, 1994.

Estrin, D., R. Govindan, J. S. Heidemann, S. Kumar, Next Century Challenges: Scalable Coordination in Sensor Networks, in Mobile Computing and Networking. 1999. p. 263-270.

Fahlman, S., Three Flavors of Parallelism, . 1982, Carnegie Mellon Universtiy: Pittsburgh, PA.

Felderman, R., A. DeSchon, D. Cohen, G. Finn, ATOMIC: A High-Speed Local Communication Architecture. Journal of High Speed Networks, 1994. 1(1): p. 1-28.

Fukuda, T., and Y. Kawauchi, Method of Autonomous Approach, Docking and Datching Between Cells for Dynamically Reconfigurable Robotic System CEBOT. Japan Soc. Mech. Eng. (JSME) Int'l Journal, 1990. 33(2): p. 263-268.

Fukuda, T., and Y. Kawauchi. Cellular robotic system (CEBOT) as one of the realization of self-organizing intelligent universal manipulator. in Proceedings of the IEEE International Conference on Robotics Automation. 1990.

Ihara, H., K. Mori, Autonomous Decentralized Computer Control Systems. IEEE Computer, 1984. 17(8): p. 57-66.

Kamimura, A., S. Murata, E. Yoshida, H. Kuraokawa, K. Tomita, S. Kokaji. Self-Reconfigurable Modular Robot -- Experiments on reconfiguration and locomotion. in Int. Conf. on Intelligent Robots and Systems. 2001. Hawaii, US.

Khoshinevis, B., R. Kovac, W.-M. Shen, and P. Will, Reconnectable Joints for Self-Reconfigurable Robots, *IROS 2001.*

Kotay, K., D. Rus, M. Vona, and C. McGray. The self-reconfiguring robotic molecule. in Proceedings of the IEEE International Conference on Robotics and Automation. 1998.

Kotay, K., D. Rus,, Locomotion Versatility through Self-reconfiguration. Robotics and Autonomous Systems, 1999. 26: p. 217--232.

Mori, K., S. Miyamoto, H. Ihara, Autonomous decentralized computer system and software structure. Computer Systems Science and Engineering, 1985. 1(1): p. 17-22.

Mori, K. Autonomous decentralized system technologies and their application to train transport operation system. in High Integrity Software Conference. 1999. Albuquerque, New Mexico.

Murata, S., E. Yoshida, H. Kurokawa, K. Tomita, S. Kokaji, Self-Repairing Mechanical Systems. Autonomous Robots, 2001. 10: p. 7-21.

Murata, S., H. Kurokawa, S. Kokaji. Self-assembling machine. in Proceedings of the IEEE International Conference on Robotics Automation. 1994.

Nilsson, M. Symmetric Docking in 2D: A Bound on Self-alignable Offsets. in Proc. IASTED'99: Robotics and Automation. 1999. Santa Barbara, California.

Pamecha, A., I. Ebert-Uphoff, G.S. Chirikjian, Useful Metrics for Modular Robot Motion Planning. IEEE Trans. on Robotics and Automation, 1997. 13(4): p. 531-545.

Roufas, K., Y. Zhang, D. Duff, M. Yim. Six degree of freedom sensing for docking using IR LED emitters and receivers. in The 7th Intl. Symp. on Experimental Robotics. 2000. Hawaii.

Rus, D., M. Vona, Crystalline Robots: Self-Reconfiguration with Compressible Unit Modules. J. of Autonomous Robots, 2001. 10(1): p. 107-124.

Rus, D., Z. Butler, K. Kotay, M. Vona, Self-Reconfiguring Robots, in ACM Communication. 2002.

Salemi, B., W.-M. Shen, P. Will. Hormone-Controlled Metamorphic Robots. in International Conference on Robotics and Automation. 2001. Seoul, Korea.

Salemi, B., W.M. Shen, P. Will. Distributed Task Selection in Chain-type Metamorphic Robots. in Intl Conference on Robotics and Automation (submitted). 2003. Taiwan.

Shen, W.-M., B. Salemi, and P. Will, Hormone-Inspired Adaptive Communication and Distributed Control for CONRO Self-Reconfigurable Robots , *IEEE Transactions on Robotics and Automation*, 18(5) 700-712, 2002.

Shen, W.-M., C.-M. Choung, P. Will, Simulating Self-Organization for Multi-Robot systems , International Conference on Intelligent and Robotic Systems, Switzerland, 2002.

Shen, W.-M. and B. Salemi, Distributed and Dynamic Task Reallocations in Robot Organization, IEEE Conference on Robotics and Automation, Washington DC, 2002.

Shen, W.M., Y. Lu, P. Will. Hormone-based Control for Self-Reconfigurable Robots. in Proceedings of International Conference on Autonomous Agents. 2000. Barcelona, Spain.

Shen, W.M., B. Salemi, and P. Will. Hormones for Self-Reconfigurable Robots. in Proceedings of the 6th International Conference on Intelligent Autonomous Systems, IOS Press, pp. 918-925, 2000.

Shen, W.M., P. Will. Docking in Self-Reconfigurable Robots. in International Conference on Intelligent Robots and Systems. 2001. Hawaii.

Sims, K. Evolving Virtual Creatures. in Computer Graphics, Annual Conference Series, (SIGGRAPH '94 Proceedings). 1994.

Simon, H.A., The Sciences of Artificial. 1996: The MIT Press.

Støy, K., W.-M. Shen, and P. Will, On the Use of Sensors in Self-Reconfigurable Robots, *In proceedings of the 7th international conference on simulation of adaptive behavior (SAB02)*, Edinburgh, UK, August 4-9, 2002.

Støy, K., W.-M. Shen, and P. Will, How to Make a Self-Reconfigurable Robot Run, *In proceedings of the 1st international joint conference on autonomous agents and multiagent systems (AAMAS'02) (to appear)*, Bologna, Italy, July 15-19, 2002.

Støy, K., W.-M. Shen, and P. Will, Global Locomotion from Local Interaction in Self-Reconfigurable Robots, *In proceedings of the 7th international conference on intelligent autonomous systems (IAS-7)*, Marina del Rey, California, USA, March 25-27, 2002.

Tomita, K., S. Murata, H. Kurokawa, E. Toshida, S. Kokaji, Self-Assembly and Self-Repair Method for a Distributed Mechanical System. IEEE Trans. on Robobtics and Automation, 1999. 15(6): p. 1035-1045.

Unsal, C., H. Kiliccote, P.K. Khosla, A modular self-reconfigurable bipartite robotic system: Implemenation and motion planning. Autonomous Robots, 2001. 10: p. 23-40.

UserManual, Knowledge Revolution Working Model 3D User's Manuals. 1997.

Vassilvitskii, S., M. Yim, J. Suh. A complete, local and parallel reconfiguration algorithm for cube style modular robots. in Intl. Conf. on Robotics and Automation. 2002. Washington, DC.

Will, M.P., A. Castaño and W.-M. Shen, Robot Modularity for Self-Reconfiguration, Proc. SPIE Sensor Fusion and Decentralized Control II, pp. 236-245, Boston, Sep. 1999.

Yim, M., D.G. Duff, K.D. Roufas. PolyBot: A Modular Reconfigurable Robot. in Proceedings of the IEEE International Conference on Robotics and Automation. 2000.

Yim, M., Locomotion with a unit-modular reconfigurable robot (Ph.D. Thesis), in Department of Mechanical Engineering. 1994, Stanford University.

Yim, M., Y. Zhang, J. Lamping, E. Mao, Distributed Control for 3D Metamorphosis. Autonomous Robots, 2001. 10: p. 41-56.

Yim, M., Y. Zhang, D. Duff, Modular Robots, in IEEE Spectrum. 2002.

Yoshida, E., S. Murata, A. Kamimura, K. Tomita, H. Kurokawa, S. Kokaji. A motion planning method for a self-reconfigurable modular robot. in Int. Conf. on Intelligent Robots and Systems. 2001. Hawaii, US.

Yoshida, E., S. Murata, K. Tomita, H. Kurokawa, and S. Kokaji. Distributed formation control of a modular mechanical system. in Proceedings of the International Conference on Intelligent Robots and Systems. 1997.

Yoshida, E., S. Murata, H. Kurokawa, K. Tomita, S. Kokaji. A distributed reconifuration method for 3-D homogeneous structure. in In Proc. IEEE/RSJ Intl. Conf. Intelligent Robotics and Systems. 1998.

7 Appendix A: Theories for Robot Self-Organization

• Shen, W.-M., C.-M. Choung, P. Will, Simulating Self-Organization for Multi-Robot Systems, International Conference on Intelligent and Robotic Systems, Switzerland, 2002.

• Shen, W.-M. and B. Salemi, Distributed and Dynamic Task Reallocations in Robot Organization, IEEE Conference on Robotics and Automation, Washington DC, 2002.

Simulating Self-Organization for Multi-Robot Systems

Wei-Min Shen[1], Cheng-Ming Chuong[2], and Peter Will[1]

[1]*Information Sciences Institute, University of Southern California, Los Angeles, USA, shen@isi.edu,will@isi.edu*
[2]*Pathology Department, University of Southern California, Los Angeles, USA, chuong@pathfinder.usc.edu*

Abstract

How do multiple robots self-organize into global patterns based on local communications and interactions? This paper describes a theoretical and simulation model called "Digital Hormone Model" (DHM) for such a self-organization task. The model is inspired by two facts: complex biological patterns are results of self-organization of homogenous cells regulated by hormone-like chemical signals [6], and distributed controls can enable self-reconfigurable robots to performance locomotion and reconfiguration [1-3]. The DHM is an integration and generalization of reaction-diffusion model [4] and stochastic cellular automata [19]. The movements of robots (or cells) in DHM are computed not by the Turing's differential equations, nor the Metropolis rule [5], but by stochastic rules that are based on the concentration of hormones in the neighboring space. Experimental results have shown that this model can produce results that match and predict the actual findings in the biological experiments of feather bud formation among uniform skin cells [6]. Furthermore, an extension of this model may be directly applicable to self-organization in multi-robot systems using simulated hormone-like signals.

1. Introduction

This paper[1] is to develop a general computational model for self-organization in multi-robot systems. In particular, we describe the Digital Hormone Model (DHM) that is generalized from an existing distributed control system for self-reconfigurable robots [1-3]. The model is inspired by the fact that many complex patterns in biological systems appear to be the results of self-organization among homogenous cells regulated by hormones, and self-organization is based on local interactions among cells rather than super-imposed and pre-determined global structures [6, 7]. The paper describes the model in detail, reports the experimental results in simulating feather buds formation among homogeneous skin cells, and finds a number of correlations between individual hormone diffusion profiles and the features of final patterns. These results match the findings in the actual biological experiments and predict cases that have yet been observed in biological experiments but consist with the expected behaviors of hormone-regulated self-organization.

[1] We are grateful that this research is in part supported by the AFOSR grants F49620-01-1-0020 and F49620-01-0441, and the DARPA contract DAAN02-98-C-4032. The second author is supported by NSF IBN 9808874 and NIH AR 42177.

2. Self-Organization in Nature

Self-organization is ubiquitous in nature. It appears in physics, chemistry, materials sciences, and others [12]. But perhaps the richest source for self-organizational phenomena is biological systems. Here, we will describe two of the most fascinating phenomena that are related to self-organization in multi-robot systems: morphallaxis and feather formation.

Morphallaxis is a process by which an organism can regenerate a part or the whole from a fragment by self-reorganization of cells without cell proliferation. This is a process of tissue reorganization observed in many lower animals following severe injury, such as bisection of the animal, and involves the breakdown and reformation of cells, movement of organs, and re-differentiation of tissues. The result is usually a smaller but complete individual, derived entirely from the tissues of part of the original animal. It is believed that such a reorganization process is the most efficient way for simple organisms to self-heal and self-regenerate. This is also extremely important for self-reconfigurable robots to perform self-repair functions. One of the most remarkable examples of morphallaxis is a type of invertebrate freshwater animal called a hydra. If a hydra is cut in half, the head end reconstitutes a new foot, while the basal portion regenerates a new hydranth with mouth and tentacles. Even if a hydra is minced and the pieces scrambled, the fragments grow together and reorganize themselves into a complete whole. How this dramatic self-healing and self-adaptation process takes place is still a mystery.

Another interesting self-organization phenomenon in biological systems is the formation of feathers. In chickens, for example, feathers are developed from skin cells during an early development stage before they hatch from the eggs. Homogeneous skin cells first aggregate and form feather buds that have approximately the same size and space distribution. The feather buds then grow into different types of feathers depending on the region of the skin. Many earlier theories believed that the periodic patterns of the feather buds are formed by sequential propagation and orchestrated by some "key" skin cells. These key cells occupy strategically critical positions on the skin. They first command their neighboring cells to form one sequence of feather buds, and then this sequence propagates to form other sequences of periodic patterns. However, recent findings in biological experiments, as

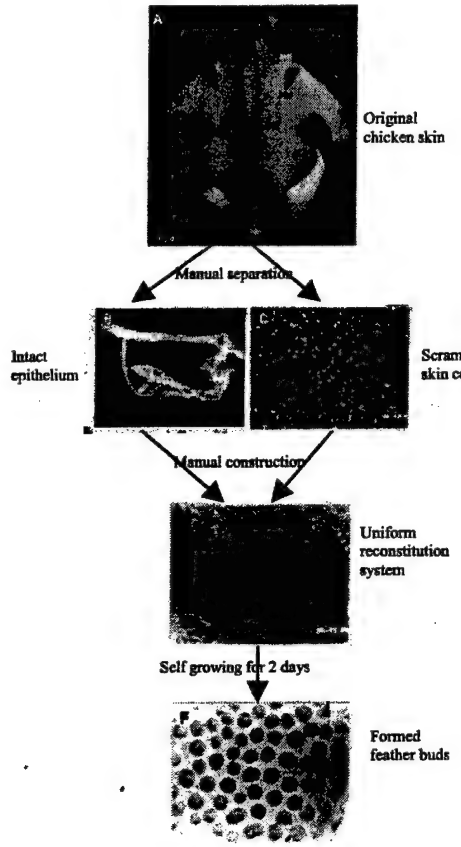

Figure 1: Self-organization in feather formation.

shown in Figure 1, have challenged these theories. Chuong and his colleagues [6, 7] first separated the embryonic chicken skin into a set of disassociated mesenchyme (i.e., skin cells before becoming feather buds) and an intact epithelium (a thin layer on which skin cells can move and aggregate). They then constructed a reconstitution system in which all mesenchymal cells are placed on the epithelium again, but scrambled and reset to an equivalent state so that they have the same probability to become primodia or interprimodia (the feather buds). Surprisingly, the cells in this reconstituted system still grow into patterns of feather buds, and such growth occurs almost simultaneously. These findings uncouple the feather bud pattern formation from the sequential propagations, and they suggest that there are no predetermined molecular addresses, and that the periodic patterning process of feather morphogenesis is likely a self-organizing process based on physical-chemical properties and reactions between homogeneous cells. For robotics research, these results indicate that it is possible for multiple robots to self-organize into interesting global patterns based on local communication and interactions without any pre-determined global patterns or structures.

During these experiments, the biologists also observed some interesting relations among the density of cell population, the individual hormone diffusion profiles, and the size and space distribution of the final patterns. In particular, they observed that while the number of formed feather buds is proportional to the cell population density, the size of the feather buds remains approximately the same regardless of different population densities. The size of the feather buds, however, is related to the diffusion profiles of the activator and inhibitor hormones secreted from the cells. If the concentration ratio of the activator to the inhibitor is high, then the final size of the feather buds will be larger than usual. If the ratio is balanced, then the size of the formed feather buds will be normal. If the ratio is low, then the size of the formed pattern will be smaller than usual. These observations are most interesting to us because they can be used as the basic criteria for evaluating computational models of self-organization in multi-robot systems.

3. Computational Models for Self-Organization

Throughout the history of science, there have been many computational models for self-organization. Perhaps one of the earliest is Turing's reaction-diffusion model [4], in which he analyzed the interplay between the diffusions of reacting species and concluded that their nonlinear interactions could lead to the formation of spatial patterns in their concentrations. Turing's model uses a set of differential equations to model the periodic pattern formation in a ring of discrete cells or continuous tissues that interact with each other through a set of chemicals he called "morphogens." Assuming that there are r = (1,...N) cells in the ring, and two morphogens X and Y among these cells, and letting the concentration of X and Y in cell r be X_r and Y_r, the cell-to-cell diffusion rate of X and Y be u and v, and the increasing rate of X and Y caused by chemical reactions be $f(X,Y)$ and $g(X, Y)$, respectively, Turing modeled the dynamics of this ring as the following set of 2N differential equations:

$$dX_r/dt = f(X_r,Y_r) + u(X_{r+1} - 2X_r + X_{r-1}),$$
$$dY_r/dt = g(X_r,Y_r) + v(Y_{r+1} - 2Y_r + Y_{r-1}).$$

By analyzing the solutions of these equations, Turing illustrated that a given ring of cells, which initially has the uniform concentration of Y and X, can self-organize through random fluctuations, chemical reactions, and diffusion, into a ring of periodic patterns in the concentration of Y. Two important conditions for Turing stability are: (1) between X and Y, one must be the inhibitor and the other activator, and (2) the inhibitor must have a greater diffusion rate than the actuator.

Turing's reaction-diffusion model was startlingly novel, and it has been supported both mathematically [13] and experimentally [14], and many applications are described in [15]. Interestingly, Witkin and Kass [16] extended the traditional reaction-diffusion systems by allowing anisotropic and spatially non-uniform diffusion, as well as multiple competing directions of diffusion. They use these models to synthesize textures with different patterns.

Cellular Automata (CA) [8, 18], especially those that have stochastic characteristics [19], are another important modeling technique for self-organization. Perhaps the most famous illustration of self-organization using CA is the game of Life, where randomly distributed cells on a space

of grids will live or die based on a set of very simple and deterministic rules. Life is a deterministic CA, but when rules of a CA have stochastic characteristics, then they could also be capable of modeling random fluctuations in the environment, and that may be a critical element in simulating interactions among many autonomous elements that perceive and react to local information in the environment. In fact, the Digital Hormone Model to be proposed here is essentially an integration of stochastic CA, reaction-diffusion models, and network-like diffusion space with dynamic topology.

Amorphous computing is another interesting technique that can be potentially useful for modeling self-organization. In an amorphous system, a large number of irregularly placed asynchronous, locally interacting computing elements are coordinated by diffusion-like messages and behave by rules and state markers. These systems have already been applied to building engineered systems for elements to organize and behave in predefined trend [9, 10]. Similar ideas may be directly applicable to self-organization [20], provided that they do not rely solely on the positional information of the self-organizing elements.

Pheromone-based multi-agent systems are also of interest as a tool for studying self-organization. There are already some experimental results [11], showing that a set of autonomous agents can use pheromones to form interesting and complex global behaviors. Such an idea shares many common design principles described here except that pheromones emphasize more diffusion than reaction.

4. The Digital Hormone Model

The Digital Hormone Model (DHM) is designed for simulating, understanding, and controlling self-organization in large-scale multi-robot systems. In this model, robots are simulated as cells that secrete hormones, and hormones diffuse and influence the behaviors of other cells. The Digital Hormone Model consists of a space (we use grids in this paper) and a set of moving cells. The term "cell" here can stand for any type of autonomous and intelligent elements, such as robots, agents, unmanned vehicles, mobile sensors, network nodes, or weapons. Among the grids, cells can live, evolve, migrate, or die as time passes. Each living cell occupies one grid at a time and a cell can secrete chemical hormones (or communication signals in general), which diffuse into its neighboring grids to influence other cells' behaviors. Hormones may have different types and diffusion functions. Two types of hormones are most common: an *activator* hormone that will encourage certain cell actions, while an *inhibitor* hormone will prohibit certain cell actions. We assume that hormones may react to each other (summation, subtraction, or modification), and may diffuse to the neighboring grids according to certain functions. Similar to the extensions used in [16], we allow anisotropic and spatially non-uniform diffusion. Cells are autonomous and intelligent robots that can react to hormones and perform actions such as *migration*, *secretion*, *differentiation*, *proliferation*, *death*, or *adhesion*.

At any given time, a cell selects and executes one or more actions according to a set of internal behavior rules. These rules can be deterministic or probabilistic. We assume that the rules are given and will not cause a cell to select conflicting actions. Given the grids, cells, hormones, actions, and rules, the DHM works as follows:

1. All cells select actions by their behavior rules;
2. All cells execute their selected actions;
3. All grids update the concentration of hormones;
4. Go to Step 1.

To illustrate the above definitions, let us consider a simple DHM_0 in Figure 2, where cells (shown as black dots in the grids) migrate on a space of N^2 grids. The space is a torus in the sense that the leftmost and rightmost columns are neighbors, and the topmost and bottommost rows are neighbors. Cells in DHM_0 have only two actions: secretion and migration, and the former is a constant action that always produces two hormones: the activator A and the inhibitor I. The diffusion rates for A and I secreted from a cell at the grid (a,b) to its surrounding grids (x,y) are characterized by Guassian distributions:

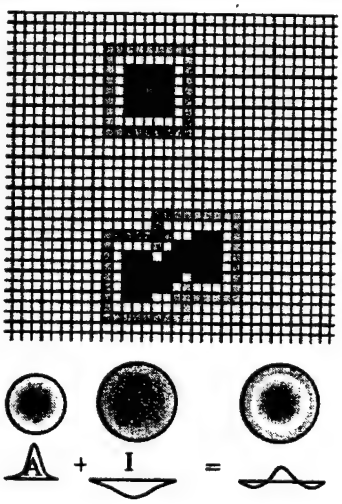

Figure 2: The simple DHM_0

$$f_A(x,y) = (2\pi\sigma^2)^{-1} \exp\{[(x-a)^2+(y-b)^2]/2\sigma^2\}$$
$$f_I(x,y) = -(2\pi\rho^2)^{-1} \exp\{[(x-a)^2+(y-b)^2]/2\rho^2\}$$

where $\sigma < \rho$ in order to satisfy Turing's stability condition. Notice that the activator A has the positive value and the inhibitor I has the negative value. Because $\sigma < \rho$, A has a sharper and narrower distribution then I. We assume that the two hormones react to each other so that the concentration of hormones in any given grid can be computed by summing up all present "A"s and "I"s in the grid. In Figure 2, we have illustrated in the grids the combined hormones around a single cell and around two nearby cells. Since the grids are discrete, the rings around the cells are shown as squares instead of circles.

In this simple model DHM_0, two simple rules govern the cell's actions. One rule states that "secrete A and I for every step", and this means that each cell secretes these hormones at every step. The second rule states that "migrate to an immediate neighbor grid based on the hormone distribution in these neighbors." More specifically, the probability for a cell to migrate to a particular neighboring grid (including the grid it is currently occupying) is proportional to the concentration of A and inversely proportional to the concentration of I in

that grid. This rule is fundamentally stochastic, so that the selection of migrating grid is non-deterministic. To implement this rule, let the hormone value in the occupying grid be h_0 and let the values in the eight immediate neighbors be h_1, h_2, h_3, h_4, h_5, h_6, h_7, and h_8, respectively. Based on their signs, these values are grouped into three groups: G1, G2, and G3, where the members in G1 all have positive values (say sum to P_{G1}), those in G2 have zero values, and those in G3 have negative values. To decide which group to migrate to, a random number x is generated in the range of (0, $100*P_{G1}+10*|G2|+|G3|$]. If $0<x<=100P_{G1}$, then the cell will migrate to G1. If $100P_{G1}<x<=100P_{G1}+10|G2|$, then the cell will migrate to G2. Otherwise, the cell will migrate to G3. The decision ensures that a cell will migrate to G1 with the highest probability, to G2 with lower probability, and to G3 with the lowest probability. After a group is selected, we then select a grid from the group with a similar procedure. For example, to select a grid from G1, a random number will be generated in the range of (0, $h_{i1}+ h_{i2}+ h_{i3}+...+ h_{i|G1|}$], where h_{ij} are individual values in G1 ($h_{ij}>0$), and a grid will be selected depending on where the number falls in the range. This ensures that grids with higher concentrations of the activator hormone will be selected with higher probabilities. To select a grid from G2, we order the grids in the group g_1, g_2, ..., $g_{|G2|}$ (note that all these grids have zero hormone values), and a random number y is generated in the range of (0, |G2|], and the grid of g_y is selected. To select a grid from G3, a random number will be generated in the range of (0, $(-h_{j1})^{-1} + (-h_{j2})^{-1} + (-h_{j3})^{-1}+...+ (-h_{j|G3|})^{-1}$], where h_{ij} are individual values in G3 ($h_{ij}<0$), and a grid will be selected depending on where the number falls in this range. This ensures that grids with lower concentrations of the inhibitor hormone will be selected with higher probabilities.

Notice that the above rule for selecting migration direction is different from the Metropolis rules used in simulated annealing [5], which first randomly selects a neighbor without considering the concentration of hormones, and then makes a go or no-go decision based on the energy difference and the current temperature. In the Digital Hormone Model, the notion of temperature is embedded in the decision rules described above. Interestingly, our experiments show that the Metropolis rule does not allow cells to converge into patterns in this model no matter what temperature is set.

Since all movements are local and synchronized, there may be a chance where multiple cells "collide" in the same grid. The collision of cells is solved in a simple manner. All cells first "virtually" move to the grids they selected. If there are multiple cells in the same grid, then the extra cells will be randomly distributed to those immediate neighboring grids that are empty. This is an environmental function, not a cellular action. But this action will ensure that no grid is hosting more than one cell at any time.

5. The Experimental Results of the DHM

Using the digital hormone models, we hope to learn valuable detailed computational knowledge about how hormones and receptors affect the result of self-organization in a large system with many autonomous elements. In particular, the initial research issues we would like to investigate are as follows:

- Will the proposed Digital Hormone Model enable cells to self-organize into patterns at all? Although self-organization has been widely studied by many different models, this is perhaps the first attempt to model mobile intelligent elements that have dynamic structure topology.
- Will the size of final patterns be invariant to the cell population density? Assuming that the cells' hormone diffusion profiles are fixed, will the results match the observations made in the biological experiments?
- Will the hormone diffusion profiles affect the size and shape of the final patterns as shown in the biological experiments?
- Will an arbitrary hormone diffusion profile enable self-organization and pattern formation? In general, how do the profiles affect the results of self-organization process?

To find solutions for these questions, we ran two sets of experiments using the simplified digital hormone model DHM_0 described above. In the first set of experiments, we set the hormone diffusion profile to approximate the standard distributions. For any single isolated cell, let the cell's n^{th} ring of neighbors be the neighboring grids at a distance of n grids away from the cell. Using this definition, we define the concentration level of the activator hormone at the cell's surrounding grids as follows: 0.16 for the 0^{th} ring (i.e., the occupying grid), 0.08 the 1^{st} ring, 0.04 the 2^{nd} ring, 0.02 the 3^{rd} ring, and 0 the 4^{th} and beyond. For the inhibitor hormone, the concentration levels for the 0^{th} through the 4^{th} rings of neighbors are: -0.05, -0.04, -0.03, -0.02, and –0.01, respectively, and 0.0 for the 5^{th} ring and beyond. Thus the combined concentration levels of hormones at the 0^{th} through 4^{th} rings are: 0.11, 0.04, 0.01, 0, and –0.01, respectively, and 0.0 for the 5^{th} ring and beyond. We assume that the concentrations of hormones secreted by a cell at grids beyond the 4^{th} ring are so insignificant that they can be practically ignored.

Given this fixed hormone diffusion profile, we have run a set of simulations on a space of 100x100 grids with different cell population densities ranging from 10% through 50%. Starting with cells randomly distributed on the grids, each simulation runs up to 1,000 action steps, and records the configuration snapshots at steps of 0, 50, 500, and 1,000. As we can see from the results in the upper part of Figure 3, cells in all simulations indeed form clusters with approximately the same size. These results demonstrate that the digital hormone model indeed enables cells to form patterns. Furthermore, the results match the

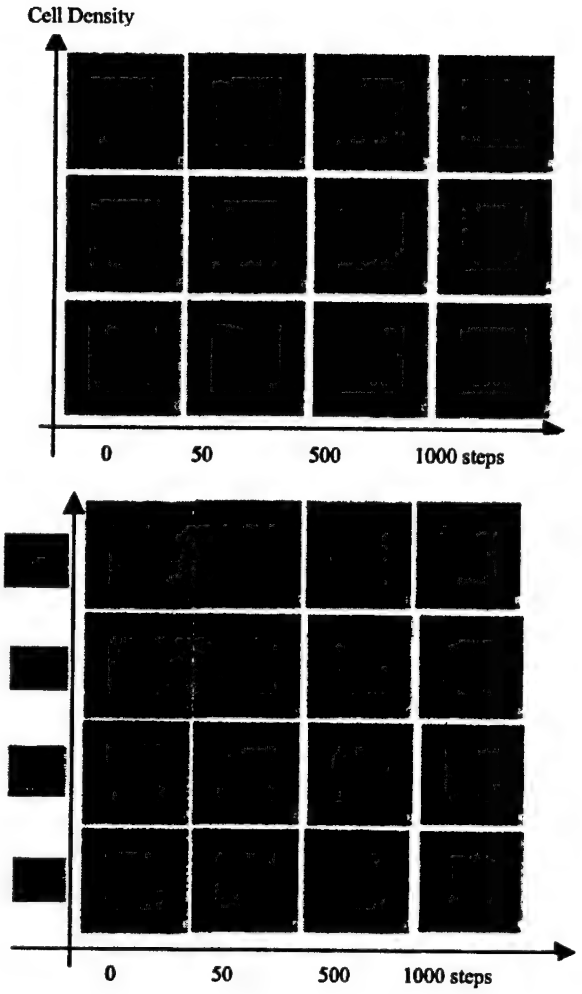

Figure 3: Two sets of experimental results on DHM_0

observations made in the biological experiments. The size of the final clusters does not change with cell population density, but the number of clusters does. Lower cell densities result in fewer final clusters, while higher densities form more clusters.

In the second set of experiments, we started with the same cell population density, but varied the hormone diffusion profiles. We wanted to observe the effects of different hormone profiles on the results of pattern formation. As we can see from the results shown in the lower part of Figure 3, when a balanced profile of activator and inhibitor is given (see the second row), the cells will form final patterns as in the first set of experiments. As the ratio of activator over inhibitor increases (see the third row), the size of final clusters also increases. These results are an exact match with the findings in the reported biological experiments [6].

When the ratio of A/I becomes so high that there are only activators and no inhibitors (see the fourth row), then the cells will form larger and larger clusters, and eventually become a single connected cluster. On the other hand, when the ratio is so low that there is only inhibitor and no activator, then the cells will never form any patterns (see the first row), regardless of how long the simulation runs. This shows that not all hormone profiles enable self-organization. These results are yet to be seen in biological experiments, but they are consistent with the principles of hormone-regulated self-organization and thus qualified as meaningful predictions of cell self-organization by hormones.

The results presented in Figure 3 not only demonstrate that the proposed digital hormone model is indeed an effective tool for simulating and analyzing self-organization phenomena, but that it is also capable of producing results that match the actual findings in the biological experiments and can predict the possible outcomes for new biological experiments. The results show that hormones play a critical role in self-organization, and they enable many autonomous elements to form globally interesting patterns based on only their local information and interactions. This provides a departure point for new hypotheses, theories, and experiments for self-organization. Since the model is mathematically adjustable, it is much more economic and efficient for scientists, including biologists, to design new experiments and to hypothesize new theories.

In addition to changing the ratio of activator and inhibitor hormones, we also have also varied the shape of hormone diffusion profiles and observe their effects on the features of the final patterns. For example, we have observed that if the profile is a narrow and long sandwich with the same orientation (the activator is in the middle and the inhibitors are on the outside), then cells will form striped patterns as shown in Figure 4. This shows that given the proper hormone diffusion profiles, the DHM will allow cells to form patterns with different shapes.

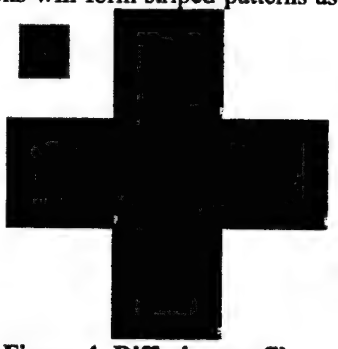

Figure 4: Diffusion profile for stripe patterns.

Furthermore, we have also experimented with different mechanisms for decision making when selecting the migration direction, including the random procedure and the Metropolis rule. Experimental results have shown that Metropolis rule does not enable cells to aggregate into groups no matter what temperature setting is used. This is a bit unexpected, but one possible reason is that Metropolis rule first randomly selects a neighbor without considering the concentration of hormones, and then makes a go or no-go decision based on probability. This does not reflect the true distribution of hormone concentration in the neighboring grids. Similarly, and as expected, the random procedure for selecting migrating directions does not produce any interesting results either.

6. Extend DHM for Multi-Robot Self-Organization

In the above discussion, we have used grids as an approximation for space in which robots can move around. Furthermore, we have assumed that grids can also perform computations to update their hormone concentration. To relax these assumptions and extend DHM for multi-robot self-organization, we must find a way to allow robots themselves to update the hormone-concentration in their surrounding space.

To simulate hormones in grids, we assume robots have wireless communications and can send each other "hormone signals" that carry the necessary information of hormone type and diffusion functions. The strengths and direction of the signal can also be used to calculate the distance and directions between robots, which are the sources of different hormones. Using the information, a multi-robot system can simulate a DHM as follows:

1. All robots select actions by their behavior rules;
2. All robots execute their selected actions;
3. All robots broadcast and receive hormone signals;
4. All robots update the concentration of hormones in their surrounding space;
5. Go to Step 1.

Furthermore, the discrete grids should be generalized into continuous space. Since we have used standard distributions for the hormone diffusion functions, this generalization is not difficult. Instead modeling the neighbors using grids, a robot's neighboring space will be modeled as a continuous circle, with directions ranging from 0.0 to 2π. Thus, robot's moving direction will also take a continuous value. To implement this extension, each robot only requires a compass to specify the continuous direction. The computation in step 3 may be expensive, but we believe a proper and inexpensive algorithm can be designed to estimate the distribution of hormones in a robot's surrounding space without demanding outrageous computational resources.

7. Conclusion

We have presented the Digital Hormone Model (DHM) as a new computational model for self-organization in multi-robot systems. As for future research directions, we will enrich the actions, rules, and hormones in this model, develop the proper algorithms for computing hormone diffusion in continuous space using robots' onboard resources, and simulate larger scale and more complex self-organization phenomena using real robots. We will also develop formal relationships between hormone diffusion profiles and the final global patterns, and investigate the strengths and limitations of DHM for self-organization in general.

References

1. Shen, W.M., B. Salemi, and P. Will. *Hormones for Self-Reconfigurable Robots.* in *Proceedings of the 6th International Conference on Intelligent Autonomous Systems.* 2000. Venice, Italy.
2. Shen, W.M., Y. Lu, P. Will. *Hormone-based Control for Self-Reconfigurable Robots.* in *Proceedings of International Conference on Autonomous Agents.* 2000. Barcelona, Spain.
3. Salemi, B., W.-M. Shen, P. Will. *Hormone-Controlled Metamorphic Robots.* in *International Conference on Robotics and Automation.* 2001. Seoul, Korea.
4. Turing, A.M., *The chemical basis of morphogenesis.* Philos. Trans. R. Soc. London B, 1952. **237**: p. 37-72.
5. Kirkpatrick, S., G.B. Sorkin, *Simulated Annealing*, in *The Handbook of Brain and Neural Networks*, M. Arbib, Editor. 1995, The MIT Press: Cambridge, MA.
6. Jiang, T.-X., Jung, H. S., Widelitz, R. B., and Chuong, C.-M., *Self organization of periodic patterns by dissociated feather mesenchymal cells and the regulation of size, number and spacing of primordia.* Development, 1999. **126**: p. 4997-5009.
7. Chuong, C.-M., Chodankar, R., Widelitz, R.B., Jiang, T. X., *Evo-Devo of Feathers and Scales: Building complex epithelial appendages.* Current Opinion in Development and Genetics., 2000. **10**: p. 449-456.
8. Gutowitz, H., ed. *Cellular Automata — Theory and Experiment.* . 1991, The MIT Press: Cambridge, MA.
9. Abelson, H., D. Allen, D. Coore, C. Hanson, G. Homsy, T.F. Knight, R. Nagpal, E. Rauch, G.J. Sussman, R. Weiss, *Amorphous Computing*, . 1999, MIT: Boston.
10. Nagpal, R., *Organizing a global coordinate system from local information on an amorphous computer*, . 1999, MIT: Boston.
11. Parunak, H.V.D., S. Brueckner. *Entropy and Self-Organization in Multi-Agent Systems.* in *International Conference on Autonomous Agents.* 2001. Montreal, Canada.
12. Walgraef, D., *Spatio-Temporal Pattern Formation.* 1996: Springer.
13. Murray, J.D., *Mathematical Biology.* 1989, New York: Springer-Verlag.
14. Ouyang, Q., H.L. Swinney, *Transition from a uniform state to hexagonal and striped Turing patterns.* Nature, 1991. **352**: p. 610-612.
15. Meinhardt, H., *Models of Biological Pattern Formation.* 1982, London: Academic Press.
16. Witkin, A., M. Kass, *Reaction-diffusion textures.* Computer Graphics (also Proc. Siggraph 91), 1991. **25**(3).
17. Ermentrout, G.G., J. Cowan, *Large scale spatially organizing activity in neural nets.* SIAM Journal of Applied Mathematics, 1980. **38**: p. 1-21.
18. Toffoli, *Cellular Automata*, in *The Handbook of Brain Science*, M. Arbib, Editor. 2000, MIT Press.
19. Lee, Y.C., S. Qian, R.D. Jones, C.W. Barnes, G.W. Flake, M.K. O'Rouke, K. Lee, H.H. Chen, G.Z. Sun, Y.G. Zhang, D. Chen, G.L. Giles, *Adaptive stochastic cellular automata: theory and experiment*, in *Cellular Automata*, H. Gutowitz, Editor. 1991, The MIT Press: Cambridge, MA. p. 159-188.
20. Wolpert, L., *Positional information and the spatial pattern of cellular differentiation.* J. for Theoretical Biology, 1969. **25**: p.1-47.

Distributed and Dynamic Task Reallocation in Robot Organizations

Wei-Min Shen and **Behnam Salemi**

Information Sciences Institute and Computer Science Department
University of Southern California
4676 Admiralty Way, Marina del Rey, CA 90292

Abstract

Task reallocation in a multi-robot organization is a process that distributes a decomposed global task to individual robots. This process must be distributed and dynamic because it relies on critical information that can only be obtained during mission execution. This paper presents a representation for this challenging problem and proposes an algorithm that allows member robots to trade tasks and responsibilities autonomously. Preliminary results show that such an algorithm can indeed improve the efficiency of organizational performance and construct a locally optimal (hill climbing) task allocation during mission execution.

1. Introduction

In an organization of multi-robots, *task reallocation* is the process of distributing a global mission task, which has been decomposed into subtasks, to the robots in the organization. In contrast to the problem of resource allocation, task reallocation emphasizes the task migration and organizational changes among robots, rather than allocation of resources.

Traditionally, task reallocations are often considered in a centralized and static setting. A single controller robot would gather and examine all the relevant information about the current organization and mission, then decide and allocate tasks for every fellow robot. The weakness of this approach is that the accuracy of this global information is not always possible to obtain and often hard to maintain. The central controller must know the decomposition of the global task and the required capabilities and resources for each subtask. It must also know every robot's available capabilities and resources. This approach also creates a fragile bottleneck in the organization. Any failures to the controller robot will paralyze the entire organization. A distributed task allocation negotiation system is presented in [1], which is based on the contract net protocol. However, this work is different form the proposed approach in this paper, since our approach considers dependencies among subtasks as a criterion for task reallocation.

Many previous approaches for task reallocation also assume a known evaluation function for measuring the quality of task allocation, and this function remains unchanged during the process of problem solving. For example, [2] solves task allocation by analyzing the evaluation function and making all necessary decisions before the problem solving starts. Similarly, [3] assumes that the values of coalitions are computable before the execution of the organization.

In real-world applications, a solution to task allocation must consider the dynamic aspects of the environment and unexpected changes in robot behaviors because a given evaluation function might be inaccurate and the capabilities and resources of robots may change. Robots in an organization must be able to negotiate without any fixed leaders and find a satisficing solution for task allocation.

Task reallocation is closely related to the problem of self-organization, where the main objective is to decide who does what and how to collaborate with others. However, self-organization is a very diverse natural phenomenon, and a coherent and general definition is still in debate. For example, [4] defines an organization as a set of problem solvers with "information and control relationships." [5] describes an organization as a set of production systems with shared variables.[6] describes an organization as a set of "routines". [2] models an organization as a task dependent structure that includes the task units to be done, the participating (universally capable) robots, an assignment of the tasks to the robots, and a workflow structure dictates the task distribution and result assembly. Most of these definitions, although they provide valuable case studies, are not operational for task reallocation during problem solving.

This paper proposes a new approach to the problem of distributed and dynamic task reallocation based on the principles of self-organization. The approach deals with the dynamic changes in the robots and in the environment by reallocating tasks based on the performance of robots. Such reallocations are made by individual robots themselves and require no central controller robot to know the global knowledge of the organization and task progress. To focus our attention on the organizational aspects of the problem, we simplify the measurement of performance by considering the costs of communication only. Our approach is similar to the bottom-up approaches for self-organization. [5]. However, we do not assume that robots have universal capabilities (i.e., every robot can handle every subtask) and the population of robots can be changed arbitrarily.

In this paper, we define a result of task reallocation as assignment of robots to a *role-graph*, and define the

process of task reallocation as the optimization of these assignments, with respects to the dynamic cost function. In a role-graph, a *role* node is a set of responsibilities (or subtasks) for the given task, and a *role-relationship* edge is a commitment between roles to communicate certain types of information (such as subtasks, solutions, actions, or data). This representation separates the role requirement from the robots' capabilities, and makes the assignment of robots to roles an essential task in organization. In this paper, we assume that an initial role-graph is given to the robots, but the robots are allowed to modify the role-graph at will. As we will see later, these modifications include trading tasks and responsibilities among robots, and such modifications can affect the structure of an organization. With this representation, task reallocation is a team-learning process for adapting a role-graph and searching an optimal robot assignment to the role-graph based on performance results during problem solving.

The rest of the paper is organized as follows. Section 2 gives a formal definition for the problem of distributed and dynamic task reallocation. Section 3 presents a solution approach based on local search with two novel heuristics for task-trading and responsibly-trading among robots. Section 4 describes the SOLO algorithm that implements the above approach. Section 5 presents the experimental results of the SOLO algorithm. Section 7 concludes the paper with future research directions.

2. Distributed and Dynamic Task Reallocation

To study the problem of distributed and dynamic task reallocation in a domain-independent fashion, it is necessary to ground the research on a rigorous computational foundation that is domain-independent and decomposable among multiple robots. Examples of such foundations include Distributed Constraint Satisfaction Problems (DCSP), Distributed Bayesian Networks, Contract Nets, and Graphical Models [7]. In this paper, we shall focus on DCSP to investigate the feasibility of the approach.

A Constraint Satisfaction Problem (CSP) is commonly defined as assigning values to a list of variables V from a respective list of domains D such that a set of constraints C over the variables is satisfied. For example, we can define an example CSP_1 as follows: $V=[x_1, x_2, x_3]$, $D=[\{1,2\}, \{2\}, \{1,2\}]$, and $C=[(x_1 \neq x_3), (x_2 \neq x_3)]$. Then a solution for CSP_1 is $(x_1,x_2,x_3)=(2, 2, 1)$. A distributed CSP is a CSP in which V, D, and C are distributed among multiple robots. A DCSP is solved if each robot solves its local portion of the CSP and the collection of all local solutions is a solution to the CSP. For instance, we can partition the above example into two parts: $V_1=[x_1,x_2]$, $D_1=[\{1,2\},\{2\}]$, $C_1=[(x_1 \neq x_3)]$ and $V_2=[x_3]$, $D_2=[\{1,2\}]$, $C_2=[(x_2 \neq x_3)]$, and assign them to two robots respectively. Then, a solution to the DCSP is $(x_1,x_2)=(2,2)$, and $(x_3)=(1)$. In the standard DCSP, however, task reallocation is not an issue because the assignment between robots and variables are given and static.

To generalize DCSP to address the problem of distributed and dynamic task reallocation, we map tasks to variables, task dependencies to constraints between variables. A task is solved if the corresponding variable is assigned a value that does not violate any involved constraints. We further introduce that (1) every task/variable has a set of required capabilities, (2) every task dependency/constraint has two responsibilities: the *supervisor* and the *subordinator*; and (3) there is a set of heterogeneous robots that collectively possess all the required capabilities. For task dependency that links two task variables, the responsibility of the supervisor is to select a value for its variable and pass the value to the subordinator. The responsibility of the subordinator is to adjust the value of its variable so that it satisfies the constraint with the supervisor's value.

Formally, the problem of distributed and dynamic task reallocation can be defined as a tuple (V, R, D, C, A), where V is a list of task/variables, R a list of required capabilities by the task/variables, D a list of value domains for the task/variables, C a set constraints, and A a set of robots with heterogeneous capabilities. The goal of this problem is to find an assignment $A \Leftrightarrow (V,C)$ that is both *complete* and *optimal*. An assignment is complete if every task and every responsibility is assigned to a *qualified* robot and no single capability is assigned to more than one task simultaneously. A robot is qualified for a task if the robot possesses the necessary capabilities required by the task. An assignment is optimal if it enables a solution to the given global problem to be found with the minimal cost. The cost of a global solution can be measured in a user-specified way. For example, it could be the total number of messages sent between robots, or the sum of computational time consumed by the participating robots. In this paper, however, we will only consider the total number of messages for communication.

To illustrate the above definitions, consider a task reallocation problem TR_1 extended from CSP_1 as follows: $V=[x_1, x_2, x_3]$, $R=[\{c_1, c_4\}, \{c_2\}, \{c_3\}]$, $D=[\{1,2\}, \{2\}, \{1,2\}]$, $C=[(x_1 \neq x_3), (x_2 \neq x_3)]$, and $A=[A_1=\{c_1, c_2, c_4\}, A_2=\{c_2, c_3, c_4\}]$. In this problem, the required capabilities for task x_1 are $\{c_1, c_4\}$, task x_2 $\{c_2\}$, task x_3 $\{c_3\}$, respectively. The robot A_1 has capabilities $\{c_1,c_2, c_4\}$ and is qualified for x_1 and x_2, and the robot A_2 has $\{c_2,c_3,c_4\}$ and is qualified for x_2 and x_3. Without loss of generality, we can simplify the problem by assuming that each task requires a unique capability so that the capability requirements for tasks can be embedded in the task variables. For each task that requires more than one capability, such as x_1 requires $\{c_1, c_4\}$, we create a new capability called $c_{1\&4}$ so that x_1 can be handled by $c_{1\&4}$. With this convention, TR_1 can be simplified as: $V=[x_1, x_2, x_3]$, $R=[\{c_{1\&4}\}, \{c_2\}, \{c_3\}]$, $D=[\{1,2\}, \{2\}, \{1,2\}]$, $C=[(x_1 \neq x_3), (x_2 \neq x_3)]$, and $A=[A_1=\{c_{1\&4}, c_2\}, A_2=\{c_2, c_3, c_4\}]$. Thus we can replace robots' capabilities by task variables as follows: $R=[\{x_1\},\{x_2\},\{x_3\}]$ and $A=[A_1=\{x_1, x_2\}, A_2=\{x_2, x_3\}]$.

With the above definition, we can enumerate all possible task allocations for a given problem by matching the qualification of robots with task variables and task dependency constraints. In our current example, there are eight possible task allocations listed in Table 1. For

example, in the solution O_1, A_1:(x_1,x_2) indicates that the robot A_1 is assigned tasks x_1 and x_2, and A_2:(x_3) indicates that the robot A_2 is assigned x_3. The responsibility assignment [$x_1 \rightarrow x_3$] means that for the task dependency between x_1 and x_3, the robot A_1 (who is assigned to x_1) is the supervisor while A_2 (who is assigned to x_3) is the subordinator. Similarly, for the task dependency between x_2 and x_3, [$x_3 \rightarrow x_2$] indicates that A_2 is the supervisor and A_1 is the subordinator.

Table 1: All possible task allocations for TR_1

	Task Assignment	Responsibility Assignment
O_1	A_1:(x_1,x_2), A_2:(x_3)	[$x_1 \rightarrow x_3 \rightarrow x_2$]
O_2	A_1:(x_1,x_2),A_2:(x_3)	[$x_1 \rightarrow x_3 \leftarrow x_2$]
O_3	A_1:(x_1,x_2),A_2:(x_3)	[$x_1 \leftarrow x_3 \rightarrow x_2$]
O_4	A_1:(x_1,x_2),A_2:(x_3)	[$x_1 \leftarrow x_3 \leftarrow x_2$]
O_5	A_1:(x_1),A_2:(x_2,x_3)	[$x_1 \rightarrow x_3 \rightarrow x_2$]
O_6	A_1:(x_1),A_2:(x_2,x_3)	[$x_1 \rightarrow x_3 \leftarrow x_2$]
O_7	A_1:(x_1),A_2:(x_2,x_3)	[$x_1 \leftarrow x_3 \rightarrow x_2$]
O_8	A_1:(x_1),A_2:(x_2,x_3)	[$x_1 \leftarrow x_3 \leftarrow x_2$]

The above task allocations can be graphically represented in a Distributed Organizational Task Network (DOTN). In a DOTN, the nodes are the task variables with required capabilities, the edges are the task dependency constraints, and the direction of an edge represents the responsibilities of the involved robots in the corresponding task dependency constraint. A DOTN is complete and optimal if the robot assignment to the elements in the DOTN (nodes and edge directions) is complete and optimal. To illustrate the representation of DOTN, Figure 1(a), 1(b), and 1(c) shows three task allocations corresponding to O_1, O_2, and O_8, respectively. Note that O_1 and O_2 have the same task assignments [A_1:(x_1,x_2),A_2:(x_3)], but different responsibility assignment for the dependency constrain [$x_2 \neq x_3$]. In O_1, A_1 is the subordinator and A_2 the supervisor, while in O_2, A_1 is the supervisor and A_2 the subordinator. In Figure 1(c), O_8 has a totally different task assignment: A_1 is assigned to x_1, and A_2 to (x_2, x_3).

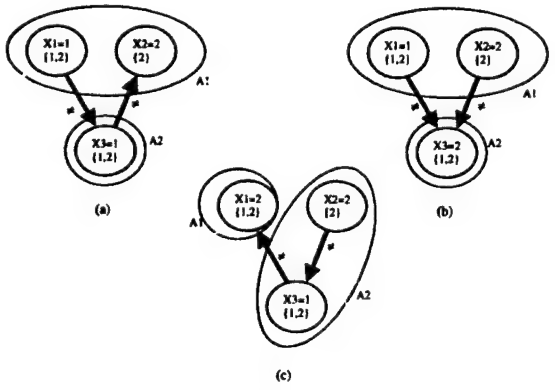

Figure 1: DOTN examples for O_1, O_2, and O_8.

Intuitively speaking, the solution O8 is the optimal task allocation because it is most likely for the robots to find a coherent solution for the global problem faced by the organization. This is because the tasks x_1 and x_2 are free from any dependency constraints so they are best to be assigned to different robots. Furthermore, for the only dependency constraint [$x_2 \neq x_3$] across the two robots, x_2 has a more restricted domain than x_3, so its assignee A_2 should be given the responsibility of supervisor. In this simple illustrative example, the objective of task reallocation is to find this optimal solution O_8 during the process of solving the global DCS problem.

3. Task Reallocation by Heuristic Search

The above definition of task allocation implies that solving the problem requires a search in an enormous space for possible assignments between robots and DOTN elements (nodes and edge directions). This search can be done either exhaustively or heuristically.

The basic idea for exhaustive search is quite simple. One can find the best task allocation by going through all possible task allocations and returning the one that has the best performance. This search is complete because it guarantees to find the optimal task allocation. But the time complexity of this search makes it impractical to use. When the environment is changed, the entire search process must be started over again.

To find a practical solution for distributed and dynamic task reallocation, we may use local search approaches to find an approximation of the best task allocation. Local search proceeds by incrementally improving the current task allocation based on the feedback of solving the global problem. To do so, we have to answer two questions: how to modify a task allocation, and when to apply these modifications so that they result in improvement. By definition, a task allocation can be modified by two actions: trade tasks among robots, and trade responsibilities among robots. We now discuss them in detail.

Trading Tasks among Robots

In a task allocation process, tasks with dependency constraints are partitioned into groups, and each group is then assigned to a qualified robot. In this context, we can classify a dependency constraint between two tasks as either *remote* (across robots) or *local* (within a robot). An optimal task allocation is the one that minimizes the remote dependencies between robots so that each robot can solve its own tasks in a relatively independent way.

How do we measure the dependencies between two tasks or two robots? The most straightforward way is to simply count the number of dependency relationships between tasks or robots. This is a static estimation and the best task allocation based on this measurement can be accomplished by analyzing the role-graph at the outset and partition the roles into groups to minimize the total number of dependency relationships between groups. However, this static approach does not consider the likelihood how a dependency can be satisfied. Such likelihood would depend on how easy to find solutions for the tasks

involved, how wide the communication bandwidth is between robots, and many other facts. Such information is not available until the problem solving process starts.

As a first attempt for dynamic task reallocation, we estimate the dependency between two tasks A and B by the number of messages exchanged between the tasks:

dependency(A,B) ≡ the_num_of_messages(A,B).

When a task *x* is traded from robot A to robot B, some local dependencies of x may become remote, while some remote dependencies of x may become local. The purpose of such task trading is to reduce the total amount of remote dependencies among all robots.

To illustrate this point, consider the task allocation in Figure 2, where five tasks t_1, t_2, t_3, t_4, and t_5 are allocated to three robots A, B, C, and the dependencies on the links are as shown. Assume that based on the qualification of the robots, only t_2 and t_3 can migrate from A to B or C.

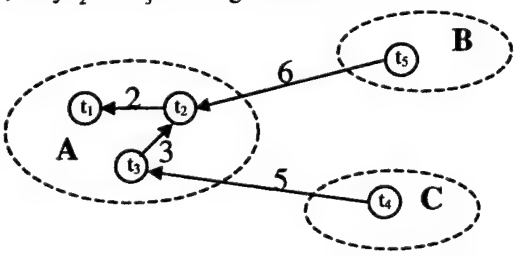

Figure 2: Trading tasks in a task allocation.

Given the above facts, the changes of remote dependencies caused by the four possible task trades can be computed as follows:

Trade t_2 from A to B: -6+3+2 = -1
Trade t_2 from A to C: 3+2 = 5
Trade t_3 from A to B: +3
Trade t_3 from A to C: -5 +3 = -2

For example, when t_2 is moved from A to B, it eliminates a remote dependency of value 6, but introduces two new remote dependencies of value 2 and 3. Among these four possibilities, the trade t3 from A to C is the most profitable modification for the current task allocation.

In general, we trade a task *x* from a robot A to another robot B if the total remote dependencies between x and tasks in B is higher than the total local dependencies of x in A. This trade of task will reduce the total remote dependencies in the entire task allocation. To facilitate this decision-making, each robot is required to record the number of messages sent and received on each task dependency.

Trading Responsibilities among Robots

Trading responsibilities between robots is another way to modify a task allocation. The basic idea is to switch the supervisor and subordinator responsibility whenever the subordinator cannot find any possible solution to its task. The motivation of this action is that by becoming a supervisor, a robot may have more freedom to choose solutions for its task, therefore more likely to find a global solution.

In DOTN, this action switches the direction of an edge and alters the direction of information flow between roles. Such changes can affect the performance of an organization because it has been demonstrated in [8] that changes in the priority order among variables can influence the rate of problem solving in DCSP. Switching responsibilities is a special case of changing priorities.

The simple protocol of switching responsibilities between two robots, however, is too limited to be effective and can run into deadlocks. Consider a situation where both the supervisor and the subordinator cannot find solutions to their tasks, then no matter how many times they switch responsibilities; the problem can never be solved because they are constrained by other neighbors.

To overcome this problem, we extend the scope of trading responsibilities from two robots to the entire neighborhood. Inspired by the priority schema in [9], we assume that every task is assigned a global priority. If two tasks are linked by a dependency constraint, then the task with the higher priority is the supervisor. Whenever a robot fails to find a solution for its task, it will switch the priority of that task with a neighboring task that has the highest priority in the neighborhood. Different from the schema in [9], this protocol does not introduce any new priorities for the tasks yet it can avoid any loop creation in DOTN.

To illustrate this protocol of trading responsibilities, assume that the tasks $[t_1,t_2,t_3,t_4,t_5]$ in Figure 2 have the priorities [1,2,3,4,5]. When t_2 fails to find a solution, it will switch its priority with t_5 because t_5 is the neighbor task that has the highest priority. Notice that after this switch, t_2 becomes the supervisor of all its neighbors, which includes t_5, t_3, and t_1.

4. The SOLO Algorithm

The process of task reallocation described above has been implemented as a new algorithm called SOLO, which is an extension of the Asynchronous Backtracking Algorithm [10] and the version that deals with multiple variables per agent [9]. Given a DCSP, the SOLO algorithm allows robots to reallocate the task variables assigned to them initially. The output of SOLO is a solution to the given DCSP and a near optimal task allocation among robots.

Four types of messages are used. (1) An *ok?* message is sent when a robot is proposing a new solution to its local task. When a robot *Ai* sends out an *ok?* message for a local task x_i, it also indicates if it is willing to give away (*giveAway?*) the task to the receiver. (2) A *nogood* message is sent when a subordinator robot finds it is impossible to find a solution for a task to satisfy all neighboring supervisors. (3) An *interested* message is sent by a qualified robot as a reply to an *ok?* message to indicate the willingness to accept the offered task. (4) A *release* message is sent by a robot to transfer a local task to an interested receiver. Among all the "interested"

```
when received (ok?, (Aj, xj, dj, priority, giveAway?)) do
  add (Aj, xj, dj, priority) to robotView;
  if (giveAway? & qualifedfor(xj) & profitToAccept(xj)) then
     send(interested, Ai, Aj, xj);
  when robotView and currentAssignments are inconsistent
     checkRobotView;
end do;

when received (nogood ,xj ,nogood) do
  add nogood to the nogoodList
  when (xk, dk, priority) is contained in the nogood
     where xk is not in the neighbors do
     add xk to neighbors,
     add (Ak, xk, dk, prority) to robotView; end do;
  checkRobotView;
end do;

when received (interested, Aj, xi) do
  if xi is in the possessedVariablesList then
     delete xi from possessedVariablesList ;
     send (release, (xi, di, priority)); end if
end do;

when received (release, (xi, di, priority)) do
  add xi to possessedVariablesList;
  annouce the new ownership of xi to the neighbors;
end do;

procedure checkRobotView
  if robotView and currentAssignments are consistent then
     for each xi that has a new value d do
        for each robot Ak that has a constraint with xi do
        giveAway?=CommCost(xi,Ak)>localCommCost(xi));
        send (ok?, (Ai, xi, d, currentPriority(xi), giveAway?));
  else select x_k from possessedVariablesList, which has the
        highest priority and violating some constraint
        with higher priority variables;
        if no value in D_k is consistent with
           robotView and currentAssignments then
           record and communicate a nogood, i.e., the subset
              of robotView and currentAssignments where
              x_k has no consistent value;
           when the obtained nogood is new do
              switch the priorities between x_k and the
inconsistent variable that has the highest priority in the nogood;
              x_k = d; where d ∈ D_k and d minimizes the
number of violations with lower priority variables;
              checkRobotView; end do;
        else x_k = d; where d ∈ D_k and d is consistent with
           robotView and currentAssignments and minimizes
           the number of violations with lower priority
           variables;
           checkRobotView;
  end if; end if;
```

Figure 3: The SOLO Algorithm

receivers, the offering robot will select the one that has the highest remote dependency with respect to the offered task. Since communication among robots are asynchronous, the offering robot cannot guarantee to wait for all interested parties so it is permissible to release the task to the first interested receiver.

Figure 3 illustrates the procedures for receiving messages such as *ok?*, *nogood*, *interested*, and *release*, and for checking local robot views. The algorithm starts with an initial task allocation determined randomly. Each robot assigns values to its local variables, and sends *ok?* messages to all related subordinator robots. After that, robots wait and respond to incoming messages. When an *ok?* message about a variable x is received, the receiver robot A will update its local view and send back an *interested* message if it is qualified to possess x and also possessing x is profitable for A. When an *interested* message for a variable x is received, the receiver robot will relinquish the variable (by deleting x from its local variable list) and replies with a *release* message. The receiver of the *release* message will add the variable to it local variables list and announce the new ownership by a set of *ok?* messages.

To illustrate the SOLO algorithm in detail, let us consider our TR1 example again. We assume that the initial task allocation is Figure 1(a). Each robot communicates these initial values via *ok?* messages. When A_1 sends an *ok?* message to A_2 for x_1's new value, it also indicates the willingness to give x_1 away because x_1 has a higher remote dependency than its local dependency. When A_2 receives this *ok?* message, it updates the *robotView* but is not interested in x_1 because the lack of qualification. After A_2 assigns a new value 2 to its local variable x_3, it sends an *ok?* message to A_1 (for A_1 is the subordinator of the constraint $x_3 \neq x_2$). This time A_1 fails to find a consistent value x_2 to satisfy the constraint of $x_3 \neq x_2$, so it performs the following actions. A_1 sends a *nogood* message $\{(x_3=2)\}$ to A_2, switches the priority value of x_2 with x_3, selects a new value 2 for x_2, sends an *ok?* message to A_2 to inform the priority switch and its willingness to give x_2 away. At this point, the task allocation becomes Figure 1(b). In this new task allocation, A_2 sends out two messages: an *interested* message to A_1 for taking x_2, and a *nogood* $\{(x_1=1),(x_2=2)\}$ message to A_1 because it fails to find a consistent value for x_3. After these messages, A_1 releases x_2 to A_2 and changes the value of x_1 to 2. At this point, all tasks are solved and the task allocation is Figure 1(c).

5. Experimental Results

We have applied the SOLO algorithm to a distributed 3-color problem. Given n variables, we first generate a random 3-color problem with $2.7n$ links (to ensure the difficulty of the problems). We then generate a set of m robots by randomly partitioning the capabilities (variables) into m even subsets. If the n/m is not an integer, then the remaining capabilities are assigned to the last robot. To make sure that robots have overlapping capabilities, we then expend each robot's capabilities by adding extra $p\%$, randomly selected different capabilities. Notice that when

p=0, every robot has unique capabilities and there is no room for trading tasks. If p=100, then all robots can trade all tasks.

Table 2 lists the results of running SOLO with different number of variables (n), robots (m), and capability overlapping (p). Each data point in the table is the average for 50 randomly generated problem instances. The initial values of the variables in these trails are determined randomly. To show the effects of task trading, we have recorded the number remote and local messages, the cycles needed to solve the problem, and the number of task trading.

Table 2: The effects of task trading

n/m	*p*=0	*p*=30	*p*=60	*p*=90
	# of remote messages			
10/4	119.0	79.3	68.1	62.0
10/8	265.1	186.7	137.4	200.8
20/8	1004.7	2394.5	1111.7	593.2
20/12	5729.7	3652.9	3038.3	1132.6
30/10	2584.0	2269.4	2490.3	2343.2
30/20	5969.4	7007.3	4236.4	348.9
50/10	2948.9	3152.1	3446.1	3310.4
100/20	6041.7	5920.6	5766.5	5718.1
n/m	*# of local messages*			
10/4	29.8	19.1	17.6	21.6
10/8	17.8	25.3	19.6	35.4
20/8	142.2	333.3	189.6	89.6
20/12	412.4	379.8	409.2	143.9
30/10	269.4	239.6	264.7	257.1
30/20	176.7	329.6	293.6	19.4
50/10	324.5	353.0	396.4	395.4
100/20	311.5	329.1	333.3	350.3
n/m	*# of cycles for solving DCSP*			
10/4	29.8	19.1	17.6	21.6
10/8	5.8	5.8	4.1	8.2
20/8	20.8	44.6	33.5	16.5
20/12	47.0	47.0	60.2	21.2
30/10	26.7	27.4	31.8	32.1
30/20	21.3	32.4	37.6	3.7
50/10	34.0	39.7	39.8	41.9
100/20	22.8	24.7	26.1	27.0
n/m	*# of task trading*			
10/4	0.0	0.7	1.1	0.5
10/8	0.0	0.0	2.5	2.0
20/8	0.0	0.1	2.8	3.9
20/12	0.0	3.7	5.0	4.9
30/10	0.0	1.3	3.6	5.8
30/20	0.0	7.0	8.0	6.6
50/10	0.0	7.0	8.7	7.4
100/20	0.0	7.2	9.9	10.8

As we can see from the results, as the overlapping capability increases, more tasks are traded between robots, less communication is needed between robots, and the rate of converge is faster (less cycles). In general, when robots have choices for what they do, task reallocation allows them to solve the problem much more quickly than fixed task allocation. We notice that communication does not reduce monotonically with the capability overlapping. In the case m/n=20/8, we see an increase of communication at 30% of overlapping, before it goes down again. Further investigation is required to determine the causes of this phenomenon.

6. Conclusion

This paper presents an approach to distributed and dynamic task reallocation in multi-robot systems. This problem is motivated by the fact that most critical information for organizational performance must be obtained during problem solving and static criteria for task allocation cannot take the dynamic information into account. The paper identifies two important heuristics for improving task reallocation and presents an initial implementation of the SOLO algorithm.

The research reported here also suggests a number of future research directions in task reallocation. In particular, more factors other than the number of messages must be considered to better estimate the dependencies between tasks and robots. Applications of the approach to real-world problems that involve robots are also necessary. An even more challenging problem is to deal with the changes in the environment where solutions to tasks are non-stationary.

7. References

1. Sandholm, T.W. *Contract Types for Satisficing Task Allocation: I Theoretical Results.* in *AAAI Spring Symposium Series.* 1998.
2. So, Y.-P., E.H. Durfee. *An organizational self-design model for organizational change.* in *National Conference on Artificial Intelligence.* 1993.
3. Shehory, O., S. Kraus. *Formation of overlapping coalitions for precedence-ordered task-execution among autonomous agents.* in *International Conference on Multiple Agent Systems.* 1996.
4. Durfee, E.H., V.R. Lesser, D.D. Corkill, *Trends in Cooperative Distributed Problem Solving.* IEEE Transactions on Knowledge and Data Engineering, 1989. **1**(1): p. 63-83.
5. Ishida, T., L. Gasser, M. Yokoo, *Organization Self-Design of Distributed Production Systems.* IEEE Transactions on Knowledge and Data Engineering, 1992. **4**(2): p. 123-134.
6. Levitt, B., J.G. March, *Organizational Learning.* Annual Review of Sociology, 1988. **14**: p. 319-340.
7. Jordan, M., *Learning in Graphical Medels.* 1998: MIT Press.
8. Armstrong, A., E. Durfee. *Dynamic prioritization of complex agents in distributed constraint satisfaction problems.* in *Proceedings of the International Joint Conference on Artificial Intelligence.* 1997.
9. Yokoo, M., E.H. Durfee, T. Ishida, K. Kuwabara, *The Distributed Constraint Satisfaction Problem: Formalization and Algorithms.* IEEE Transactions on Knowledge and Data Engineering, 1998. **10**(5): p. 673-685.
10. Yokoo, M., K. Hirayama. *Distributed Constraint Satisfaction Algorithm for Complex Local Problems.* in *International Conference for Multiple Agent Systems.* 1998.

8 Appendix B: Role-Based Control for Self-Reconfigurable Robots

- K. Støy, W.-M. Shen, and P. Will, On the Use of Sensors in Self-Reconfigurable Robots, *In proceedings of the 7th international conference on simulation of adaptive behavior (SAB02)*, Edinburgh, UK, August 4-9, 2002.
- K. Støy, W.-M. Shen, and P. Will, How to Make a Self-Reconfigurable Robot Run, *In proceedings of the 1st international joint conference on autonomous agents and multiagent systems (AAMAS'02)*, Bologna, Italy, July 15-19, 2002.
- K. Støy, W.-M. Shen, and P. Will, Global Locomotion from Local Interaction in Self-Reconfigurable Robots, *In proceedings of the 7th international conference on intelligent autonomous systems (IAS-7)*, Marina del Rey, California, USA, March 25-27, 2002.

On the Use of Sensors in Self-Reconfigurable Robots

K. Støy* **W.-M. Shen**** **P. Will****

*The Maersk Institute,
University of Southern Denmark,
Campusvej 55,
DK-5230 Odense M,
Denmark
kaspers@mip.sdu.dk

**Information Sciences Institute
University of Southern California
4676 Admiralty way,
Marina del Rey, CA 90292,
USA
{shen,will}@isi.edu

Abstract

In this paper we investigate the use of sensors in self-reconfigurable robots. We review several physically realized self-reconfigurable robots and conclude that little attention has been paid to the use of sensors. This is unfortunate since sensors can provide essential feedback that can be used to guide self-reconfiguration and control. In the systems that do use sensor feedback, the feedback is used locally on each module. However we identify a need in some situations to use sensor feedback globally. We therefore propose an approach where raw sensor values are abstracted and propagated to all modules. The sensor values are abstracted differently depending on the position of the producing sensor on the robot. We combine this approach with role based control, a control method for self-reconfigurable robots that we have developed earlier. We demonstrate that by combing these two approaches it is possible to make a self-reconfigurable robot consisting of six modules walk and avoid obstacles. However the reaction time of the robot is slow and therefore we discus possible ways of reducing the reaction time.

1 Introduction

In this paper we focus on self-reconfigurable robots built from a possibly large number of physically independent modules. The modules of these robots are able to connect and disconnect autonomously and can have sensors, actuators, a processor, a power source, and a communication system.

Self-reconfigurable robots have several advantages over traditional fixed shaped robots. 1) The modules can connect in many ways making it possible for the same robotic system to solve a range of tasks. This is useful in scenarios where it is undesirable to build a special purpose robot for each task. 2) Self-reconfigurable robots can adapt to the environment and change shape as needed. This could for instance be useful in a retrieval scenario where the robot has to snake its way through the rubble of a collapsed building and at some point change shape to recover an object from the rubble. 3) Since the robot is build from many independent modules it can be robust to module failures. If one module is defect it can be ejected from the system and the robot can still perform its task. 4) Self-reconfigurable robots are built from many identical modules. These modules can be mass produced and therefore the cost can be kept low despite their complexity. The advantages of self-reconfigurable robots can be summarized as: versatility, adaptability, robustness, and cheap production compared to the complexity and versatility of the resulting robot.

2 Research Challenges

In order to realize the potential of self-reconfigurable robots a number of research challenges have to met. There are interesting challenges to be met both in hardware and software.

2.1 Hardware Issues

In hardware some of the fundamental questions are: Does each module need computation power on-board? Where from do the modules get their energy? What kind of sensors are needed? What kind of communication system does the modules need? Several systems have been build to try to answer these questions and their properties are summarized in Table 1.

In Table 1 the systems are approximately sorted in order of increasing autonomy. It can be can seen from the table that very few of these systems are fully autonomous. This is not encouraging, because in order to realize the potential of self-reconfigurable robots it is important that they are autonomous. One important step toward achieving autonomy is to understand how to use sensors to make the robot able to sense and react to its environment. In this paper we take one small step toward understanding this.

Robot	CPU on-board	power on-board	sensors used for	Communication	Reference
JHU Hexagonal	yes[1]	no	n/a	n/a	(Pamecha et al., 1996)
JHU Rectangular	yes	no	n/a	n/a	(Pamecha et al., 1996)
MEL 3d unit	no	no	joint position	serial w. host	(Murata et al., 1998)
RIKEN vertical	no	yes	none	radio w. host	(Hosokawa et al., 1998)
PolyPod	yes	no	force/torque joint position	bus	(Yim, 1994)
Xerox PARC PolyBot	yes	no	joint position docking aid	CANbus	(Yim et al., 2000)
MEL 2000	yes[1]	no	none	serial bus with IDs[2]	(Murata et al., 2000)
MEL fractum	yes	no	none	inter-unit optical	(Murata et al., 1994)
Dartmouth molecule	yes[1]	no	not reported	serial w. host	(Kotay et al., 1998)
CMU ICES-cube	yes[1]	yes	joint position	serial w. host	(Ünsal and Khosla, 2000)
Dartmouth crystalline	yes	yes	joint position	sync. signal f. host	(Rus and Vona, 2000)
USC CONRO	yes	yes	docking aid	inter-unit optical	(Shen et al., 2000a)
CEBOT	yes	yes	docking & obst. avoid.	on-board	(Fukuda and Nakagawa, 1990)

Table 1: Overview of physically realized self-configurable robots.

2.2 Software Issues

There are two main categories of approaches to the control of self-reconfigurable robot: centralized control and distributed control.

In centralized control a central host dictates the actions of each module (Castano et al., 2000b). The advantage of this approach is that it is potentially easier for a controller to handle the complexity of the system when global knowledge is available. The disadvantage is that when the number of modules increase the host becomes the bottleneck. This scalability problem can be reduced by having modules do low-level control and having the central host coordinate the actions of the modules (Yim, 1994). The problem can also be side stepped by having a reconfiguration sequence computed off-line and afterward downloaded into the modules (Rus and Vona, 2001). The central host in this situation works as a conductor telling the modules how far they are in their action sequences. Calculating the reconfiguration sequence off-line unfortunately has the disadvantage that no adaptation can be made when the system is online. Furthermore systems based on centralized control are not robust since they all rely on a single host to function.

In order to address these problems distributed control has been used. Two classes of distributed control systems exist: synchronous and asynchronous distributed control systems. In synchronous control the strict control of the system is maintained, but the problem of robustness is removed. In synchronous control a distributed synchronization algorithm can be used to synchronize the actions of the modules connected in a low bandwidth network. Basically the synchronization signal that before came from a central host now is produced using a distributed synchronization algorithm. The hormone based algorithms developed by Shen, Salemi, and others are examples of this approach (Shen et al., 2000a, Shen et al., 2000b, Salemi et al., 2001). This solves the problem of robustness: there is no central host and the system still works even though a module is taken out. However insisting that all the modules should be strictly synchronized has a cost in terms of efficiency.

Another approach to distributed control is asynchronous distributed control. In synchronous control the modules were considered part of the whole. In asynchronous control each module is considered the whole. A module can be combined with more modules to make a bigger whole, but it in itself represents the whole. In this approach the asynchronous nature of the system is embraced and the idea of having a strictly controlled system is abandoned. The autonomy of each module is increased and the focus is on the local interaction between modules (Murata et al., 1994). The problem in asynchronous control is how to get coherent global behavior to emerge out of local interactions between many modules. The advantage of these systems

[1] but controlled off-board.

[2] local communication under development.

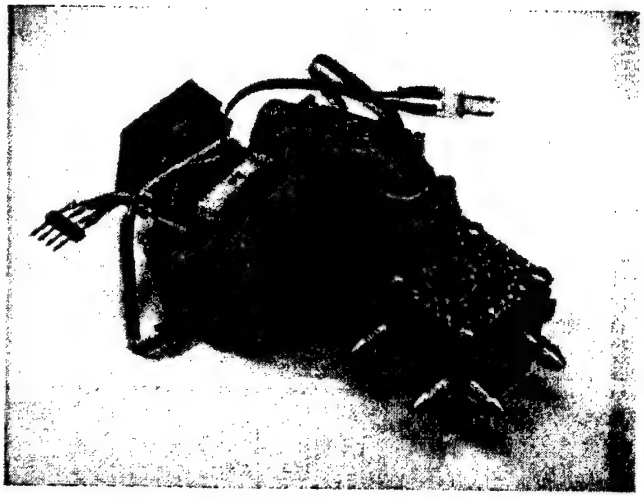

Figure 1: A CONRO module.

is that since all information is handled locally the systems scale. These methods take their inspiration from multi-agent systems (Bojinov et al., 2000b), and cellular automata (Hosokawa et al., 1998, Butler et al., 2001). Their similarities to minimalist collective robotics are also apparent see for instance (Beckers et al., 1994, Beckers et al., 2000, Støy, 2001).

In *role based control* which we will present in detail in Section 5 we combine some of the ideas from synchronous and asynchronous control. We acknowledge the need for each module to only use local information to insure scalability and robustness. However in situations where the modules are cooperating tightly, for instance to produce a locomotion gait, there is also a need to keep the modules synchronized. In role based control synchronization is achieved by having modules synchronize with neighbors from time to time. Over time this leads the entire system to be synchronized. This way the synchronization mechanism is decoupled from the control of the individual module and a robust, scalable, and synchronized system is the result.

3 Sensors and Self-Reconfigurable Robots

In research on sensor fusion there has been some work on how to combine and abstract sensor values into logical (Henderson and Shilcrat, 1984) and virtual sensors (Rowland and Nicholls, 1989). This work has been further extended with a commanding sensor type (Dekhil et al., 1996). The focus was on improving fault-tolerance of sensor systems and aiding development by making the sensor systems modular (Hardy and Ahmad, 1999). These ideas are relevant to the use of sensors in self-reconfigurable robots. However using sensors in self-reconfigurable robots is different because of the unique features of self-reconfigurable robots.

In a self-reconfigurable robot it can not be assumed that the position of the sensor is fixed. It can be moved through reconfiguration or maybe just by movements of the modules. This means that we need to understand how to extract meaningful sensor data from a network of sensors connected in time-varying ways. The previously proposed approaches also mainly deal with one consumer of the sensor date. If distributed control is employed there are many controllers that act on the sensor data. This means that system should be able to deal with inconsistent sensor data.

In distributed systems the problem of many decision makers can be dealt with in two ways. One way is to consider the modules independent and always handle sensor information locally. That is, on the module that receives the sensor input. Butler et al (Butler et al., 2001) have in simulation made a system where each module is a cellular automaton that reacts to its local configuration and surrounding obstacles. Using seven rules the modules are able to role over and across each other to produce "water-flow" like locomotion through an environment with obstacles. A similar idea was explored earlier on a real robot by Hosokawa et al (Hosokawa et al., 1998). Another approach explored by Bojinov et al (Bojinov et al., 2000a, Bojinov et al., 2000b) is to have the structure of the robot grow from seed modules. The growth is accomplished by having the seed module attract spare modules to a specific position with respect to the seed by using a virtual scent. When a spare module reaches that position the old seed module stops being a seed and the newly arrived module becomes the seed. The behavior of the seed module is controlled based on events it can sense in the environment. In these approaches the modules are decoupled in the sense that the modules only interact through stigmergy (Beckers et al., 1994).

In some systems the modules are highly coupled and sensor information can not always be handled locally: a sensor input might have effects in other modules than in the one in which it originated. This raises a fundamental questions which is the main focus of this paper: how do we distribute sensor information in order for it to arrive at the modules that need it? In this paper we present a system where sensor information is abstracted and propagated to all modules in the systems. Each module in the system then independently decides what action to take based on these propagated sensor values. Our use of sensors is inspired by the use of sensors in behavior based robotics (Arkin, 1998, Matarić, 1997) where sensors are used directly to control motors and not to build a geometrical model. We combine this communication system with role based control which we have developed earlier for the control of self-reconfigurable robots (Støy et al., 2002a, Støy et al., 2002b).

4 The CONRO module

Before describing this approach in more detail we will describe the CONRO self-reconfigurable robot. The CONRO modules were developed at University of Southern California's Information Sciences Institute (Castano et al., 2000a, Khoshnevis et al., 2001) (see figure 1). The modules are roughly shaped as rectangular boxes measuring 10cm x 4.5cm x 4.5cm and weigh 100grams. The modules have a female connector located at one end facing south and three male connectors located at the other end facing east, west, and north. Each connector has an infra-red transmitter and receiver used for local communication and sensing. The modules have two controllable degrees of freedom: pitch (up and down) and yaw (side to side). Processing is taken care of by an onboard Basic Stamp 2 processor. The modules have onboard batteries, but these do not supply enough power for the experiments reported here and therefore the modules are powered through cables. For the experiments reported here we also equipped the modules with flex sensors. The flex sensors are mounted on small circuit boards and Velcro is used to attach them to the modules. This mounting strategy is not very robust, but it is very flexible making it easy to experiment with different sensor morphologies. The flex sensors are 11cm long. Their resistance increase as they are bent and can therefore be used for rich tactile sensing. Refer to http://www.isi.edu/conro for more details and for videos of the experiments reported later in this paper.

5 Short Introduction to Role Based Control

In previous work we have introduced role based control. Role based control is a simple minimalist approach to the control of self-reconfigurable robots. We have shown earlier how this control method can be applied to chain and tree configurations to implement caterpillar like locomotion, locomotion similar to that of a sidewinding snake, and rolling track locomotion (Støy et al., 2002a). We have also used the method to make the CONRO robot configured as a hexapod and a quadruped robot walk (Støy et al., 2002b). Here we summarize role based control.

5.1 A Role

A role r consists of three components. The first component is a function $A(t)$ that specifies the joint angles of a module given an integer $t \epsilon [0:T]$. Where T is the period of the motion and the second component that needs to be specified. The third component is a set of delays D. A delay $d_i \epsilon D$ specifies the delay between the child connected to connector i and the parent. That is, if the parent is at step $t_{parent} = t_1$ the child is at $t_{child} = (T + t_1 - d_i)\, modulus\, T$. Below is some examples of roles used in the quadruped robot shown in Figure 4. The spine modules play the spine role:

$$A(t, spine) = \begin{cases} pitch(t) = 0^\circ \\ yaw(t) = 25^\circ \cos(\frac{2\pi}{T}t + \pi) \end{cases} \quad (1)$$

$$d_{east} = \frac{T}{4} \quad (2)$$

$$d_{south} = \frac{2T}{4} \quad (3)$$

$$d_{west} = \frac{3T}{4} \quad (4)$$

$$T = 180 \quad (5)$$

The legs play the forward role below or the backward role where t is replaced by $2\pi - t$ giving the same motion, but in the opposite direction.

$$A(t, forward) = \begin{cases} pitch(t) = 35^\circ \cos(\frac{2\pi}{T}t) - 55^\circ \\ yaw(t) = 40^\circ \sin(\frac{2\pi}{T}t) \end{cases} \quad (6)$$

$$T = 180 \quad (7)$$

5.2 Playing a Role

The algorithm that we now will describe is used to make a module play a role. However first some assumptions need to be made: a parent connector is specified and the remaining connectors are considered child connectors, connections can only be made between a parent connector and a child connector. Furthermore we assume that there are no loops in the configuration. These assumptions limit the configurations the algorithm can handle to tree configurations.

The algorithm has two components. One component makes sure that the actions are executed as specified in the role definition. This component also is responsible for synchronization with neighboring modules. The second component is discovering what role the module should play in case it can play more than one role. What role to play is discovered based on information propagated down from the parent and the local configuration.

The role playing component is visualized in Figure 2. The algorithm starts by setting $t = 0$ and continues to the main loop. Here the algorithm first checks if t is equal to the delay specified for each connector. In case t equals one of these delays d_i a signal is send through the corresponding child connector i. If the module has received a signal from its parent, t is reset. After that the joints are moved to the position described by $A(t)$. Finally t is incremented unless a period has been completed in which case t is reset and another iteration of the loop is initiated.

In some situations it is desirable for a module to be able to play different roles depending on its location in

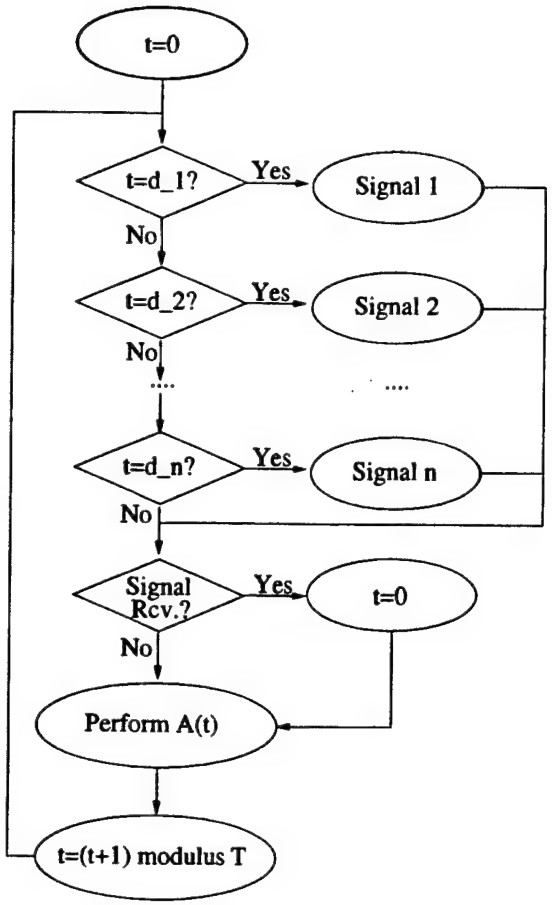

Figure 2: Visualization of the role playing part of the algorithm. See section 5.2 for an explanation.

the configuration tree. This is taken care of by the role selection component of the algorithm. The role can be selected based on the local configuration. Therefore every time a signal has successfully been sent through a child connector meaning that a child module is connected to that connector there is a check to see if the role should be changed because of that. In the basic algorithm the parent signaled the children to keep them synchronized the parent now sends a message which is a function of the parents role and the connector to which the child is connected. These two algorithmic components combined are shown in pseudo code in figure 3.

We have used this algorithm to make the CONRO self-reconfigurable robot walk. In the walker two modules are connected to form a spine. One module is connected on each side of the spine modules (see Figure 4). In this configuration the modules can play three different roles: east leg, west leg, and spine. A modules decide which role to play using the following rules: if communication to the sides (to the legs) is successful the module plays the spine role. It plays the role of a west leg if its parent is a spine module and it received the synchronization mes-

```
r = <start role>
t = 0
while(1)
  if (t=d(r)_1) then
    <send message M(r,1) to child connector 1>
    <update r>
  endif
  ...
  if (t=d(r)_n) then
    <send message M(r,n) to child connector n>
    <update r>
  endif

  if <message m received from parent connector> then
    t=0
    <update r based on m>
  endif

  <perform action A(r,t)>
  t = (t+1) modulus T(r)
endwhile
```

Figure 3: The algorithm used to play multiple roles. Refer to section 5.2 for further explanation.

sage through the west connector of the parent module. A module plays east leg if the synchronization message was send through the east connector.

Role based control is an example of an synchronous control method that does not insist on all the modules being synchronized at each step, but achieves this over time. This makes a role based system like the walker efficient because the modules work independently most of the time and only occasionally share information and synchronization information with neighboring modules. In this system all modules run identical programs and therefore modules can be interchanged and switch roles accordingly resulting in a very robust system. For more information on this system and role based control refer to (Støy et al., 2002b).

We have now summarized how role based control can be used to make the CONRO self-reconfigurable robot walk. However the system is open-looped in the sense that no sensor input from the environment is used in the control. Therefore we want to extend role based control to include sensor feedback. This is the subject of the following sections.

6 Role Based Control using Propagated Sensor Information

The CONRO self-reconfigurable robot is now configured into a quadruped robot as shown in Figure 4. Two flex sensors are attached to the front spine module and one is attached to each of the front legs. We now want to use feedback from these sensors to make the robot steer away from obstacles. In general the direction of loco-

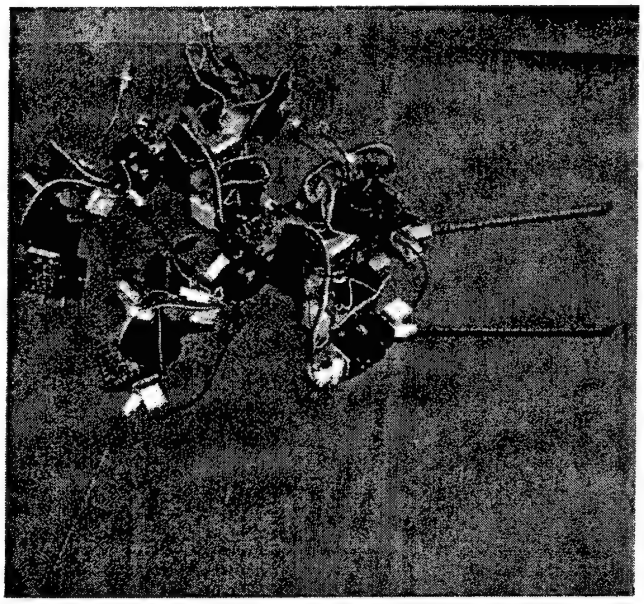

Figure 4: The CONRO robot in a walker configuration. The spine is made from two modules and the legs are made from one module each.

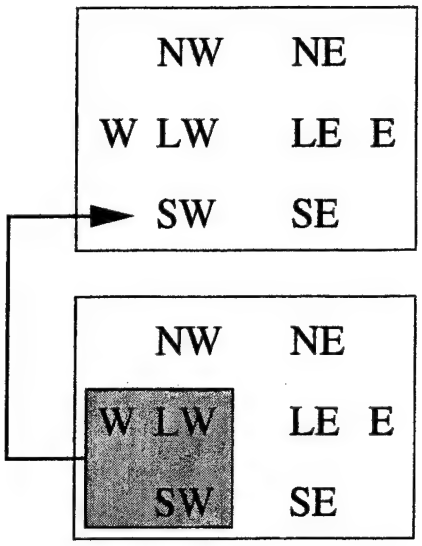

Figure 5: This figure shows how sensor values are propagated north from one spine module to the next. The south module (bottom) sums up the variables in the gray box and sends the sum to the north module (top). The north module receives this sum and write it in the variable indicated by the arrow.

motion can be changed in two ways: the motion of the legs can be biased so legs on the side pointing away from the obstacle take shorter steps and those on the other take longer steps. This approach will enable the robot to make soft turns away from obstacles. An alternative is to have legs on the side pointing away from the obstacle to move backwards and thus producing a turning on the spot motion. We found in initial experiments that the sensor based bias of the locomotion pattern does not produce a sharp enough turn to avoid obstacles. Therefore we decided to implement roles that make it possible for the robot to turn on the spot.

The goal is to make the quadruped robot turn on the spot away from an obstacle detected using one of the four flex sensors. Below we describe how this is achieved. A module has up to two flex sensors mounted: the front legs have one each, the front spine module has two, and the rest zero. The modules continuously sample these sensors and write the analog value into a local variable. What variable dependents on the position of the sensor. If the flex sensor is pointing toward the east the sensor value is written in a variable named local east (LE) and if it points west in local west (LW). If there are no sensors attached these variables contain zero. Each module has an additional six variables: northeast (NE), northwest (NW), east (E), west (W), southeast (SE), southwest (SW). These variables represent the sensor activity in the direction indicated by the names. For instance if all the west variables including local west are added up it will give the sum of the sensor activity on the west side of the robot. The same is true for the east values.

We will now describe how the sensor values are propagated in the system to produce the contents of the variables as it is described informally above. When a spine module sends sensor information to a module connected to its north connector it works as follows. The south module adds up the variables west, southwest, and local west and sends the sum to the north module. The sum is received by the north module and is written in the southwest variable. Note that this satisfy the invariant that the southwest variable of the north module now contains the sum of the sensor activity to the southwest. This mechanism is summarized in figure 5. At the same time the sum of the east variables is propagated and written in the southeast variable of the north module.

This mechanism will sum up the sensor inputs all along the spine and the northern most module will have information about the sensor activity to the southeast and southwest. However we want all the modules to have information of sensor activity in all directions therefore a similar, but independent mechanism is also propagating sensor activity southward. These two propagation mechanisms are summarized in figure 6.

All the spine modules now have information about sensor activity along the spine. For instance the modules can sum the northwest, local west, and southwest variables to find the sensor activity on the west side of the robot. This is not enough in our situation, because there are also two legs attached to the spine modules. Therefore sensor information should also be propagated from and to them. In our setup an east leg can only have one piece of sensor information that the spine does not have:

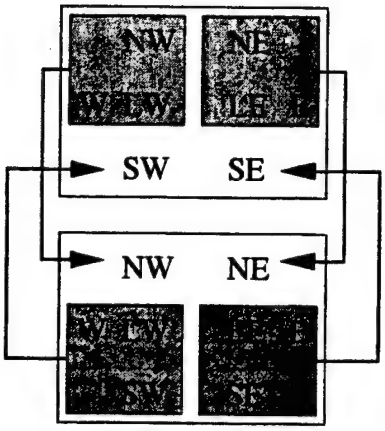

Figure 6: This figure shows how sensor values are exchanged between two spine modules. The modules sum the variables in the gray boxes and send these two values to the other module. The receiving module then writes these values in the variables as indicated by the arrows.

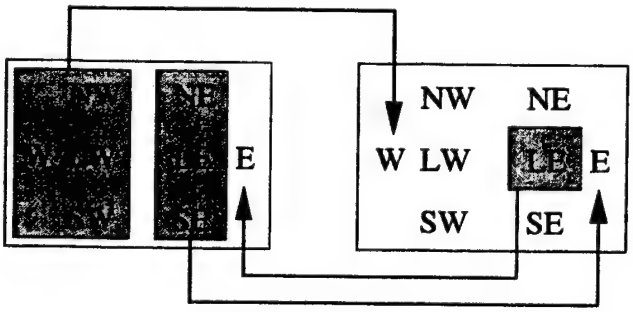

Figure 7: This figure show how sensor values are exchanged between a spine module (left) and an east leg module (right). The modules adds up the numbers in the gray boxes and send these values to the other module. The receiving module then write these values in the variables as as indicated by the arrows.

the value of the sensor connected to that leg. Therefore the local east value is propagated from the leg to the spine. The leg on the other hand receives the sum of all the sensor activity on the west side of the robot and writes that in the west variable. The sum of the variables northeast, local east, and southeast of the spine is written in the east variable of the leg. This is summarized in figure 7.

In role based control synchronization information from the parent module is sent to child modules each period of locomotion. The sensor information is also exchanged at this time as described above. How the sensor information is exchanged depends on what role the module plays. This is decided by the role playing algorithm. Therefore modules can still be exchanged and the system will continue to function if the sensors are placed correctly.

All the modules now have access to global sensor information and can make their decisions based on this information. In order to make a decision we sum the variables for the west and east side of the robot to have a measure of the activity on each side of the robot. A leg then decides to move backward if the sensor activity on the other side of the robot is above a small threshold and higher than the sensor activity on the leg's side. Otherwise it will move forward.

7 Results

First we will note some general properties of the system. One step of the robot corresponding to one period of locomotion takes two seconds. The step length is 15cm. Note that a step is quite long compared to the length of a module (10cm). The long steps are achieved by actively using the spine to make the steps longer. The robot achieves a speed of 7.5cm/second.

In four separate experiments the robot was placed so it approached an obstacle from four different angles. These experiments were videotaped using an overhead camera. We then manually analyzed the tape and for every two seconds recorded the position of the front end of the robot, the rear end, and whether a flex sensor was touching the obstacle. The results of this analysis can be seen in Figure 8.

The sensor values are exchanged when modules synchronize. We know the spine synchronizes with the east leg at $T/4$, the spine module to the south at $2T/4$, and the west leg at $3T/4$. We can use this information to calculate upper and lower bounds on communication delays. In the worst case where the sensor change happens just after synchronization it takes 2 periods to get sensor information from a front leg to the rear leg on the other side. In the best case where the sensor change happens just before synchronization it takes 1 period. This means that the whole system has a reaction time between two and four seconds or a reaction distance of 15cm to 30cm. Note that the reaction time is much better for the front legs. We can see these slow reaction times in Figure 8. The robot can only successfully avoid the obstacle when it approaches at an angle. In trial number four where the robot does not have time to react it bumps into the obstacle. This also explains why we decided to implement the turning on the spot behavior.

8 Discussion

If we look at how our approach can be used in general. We note that there are two things that make out system work. 1) The sensor data is abstracted based on the sensors position in a way that is useful for the receiving modules. 2) The abstracted sensor values are propagated at a constant slow rate to all modules.

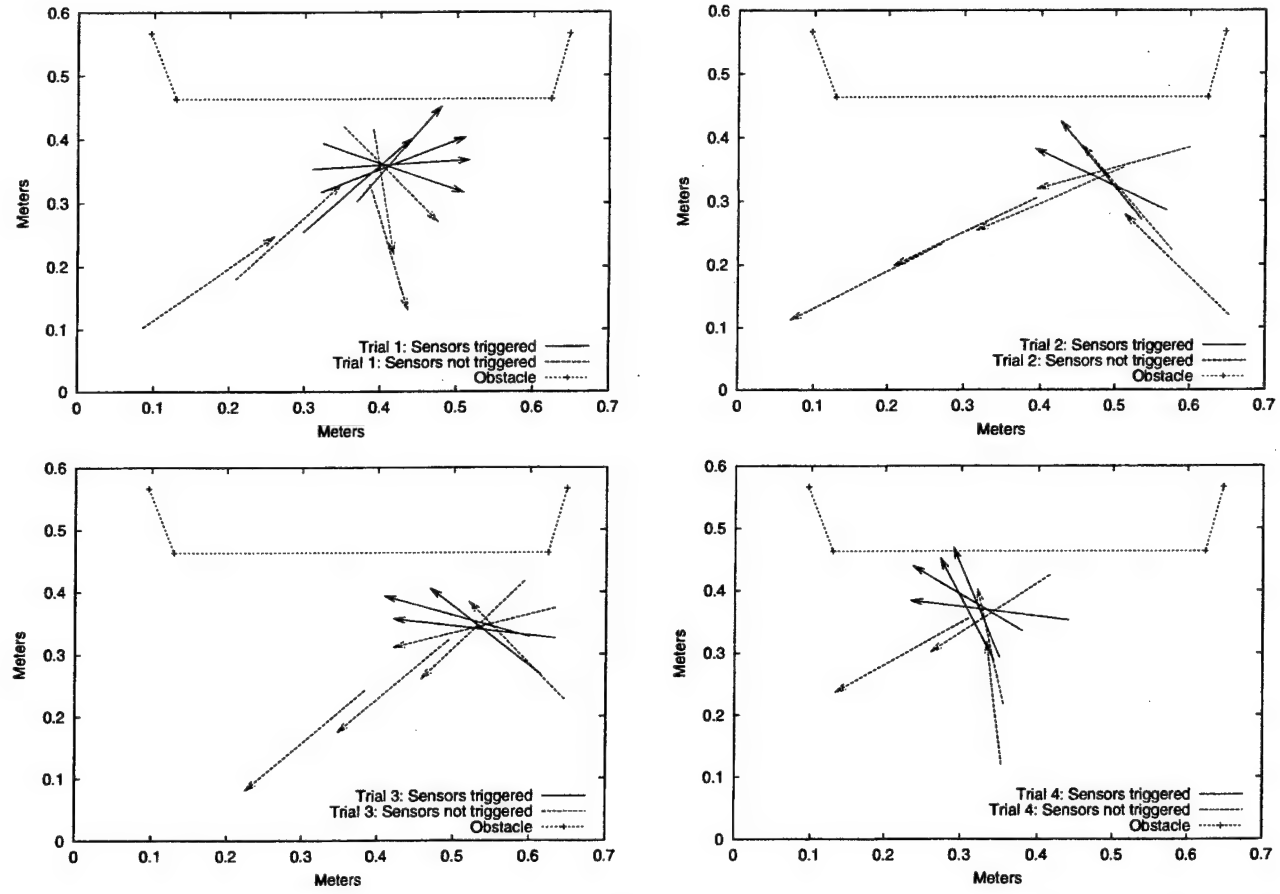

Figure 8: These figures show the robot approaching an obstacle, turning on the spot, and moving away. The arrows represent the positions of the front end and rear end of the robot recorded every 2seconds. The direction of the arrow shows the direction of movement. A solid arrow indicate that a flex sensor was triggered in that time step. A dashed that none were triggered.

In order to keep the amount of communication manageable we abstract sensor values to maintain an estimate in each module of the sensor activity to the left and right of the robot. It is possible for all the modules to agree on left and right, because of the properties of the CONRO hardware. The modules can only be connected in a tree structure (with one loop) and be connected in four ways and therefore the transformation of direction from module to module is easy. In general it seems to be important to have some relative position information about the sensor with respect to the acting module. This means that in systems where the relative position of two connected modules can be found it is possible to abstract the sensor information in a useful way. This is also what makes this approach different from previous work on sensor fusion. The position of the sensor is variable and this is in our case handled by the abstraction mechanism.

Sensor values flow around in our system at a constant slow rate. This rate could be increased significantly to reduce the reaction time. The problem of doing this is that our modules have limited resources and therefore if time is spent on communication less time can be spent on control of the motors resulting in a decrease in speed. Therefore in order to decrease the reaction time of the system without sacrificing speed we need to use less communication and achieve a shorter reaction time at the same time. One solution would be to have the module monitor a sensor and if it goes above a certain threshold it can be propagated. When the sensor later drops below the threshold another message can be propagated. This might improve the response time of the system, because when communication takes place only when it is needed it can be made efficient.

Another orthogonal way to decrease the amount of communication would be to only propagate sensor information to the modules that need it. For instance in the walker sensor information from one side could be propagated to the other side. In this way the sensors on the left control the legs on the right and the other way around. In general directed diffusion (Intanagonwiwat et al., 2000) could be used for this. In directed diffusion information is propagated from the producer to the consumer through networks with a time varying configuration. In this framework it is possible

for the consumer to show his interest in a specific kind of data and have that routed to it from the producer.

9 Summary

In a self-reconfigurable robot sensors can be used in two ways. One way is to use the sensor information locally: sensors can be used to bias the motions of modules to produce the desired global behavior. The advantage of this is efficiency since the sensor values do not need to be communicated anywhere.

In some situations it is not possible to handle all sensor information locally. Therefore another way to handle sensor information has been presented. We have experimented with an approach where sensor information is abstracted based on the sensors position on the robot. This abstracted sensor information then flows from the modules that produce the information to all the modules of the system. An important aspect of this system is that the acting modules have abstract information about the position where the sensor value originated. We have shown that by combining this communication mechanism with role based control we were able to make a six module self-reconfigurable robot walk and avoid an obstacle.

Acknowledgments

This research is funded under the DARPA contract DAAN02-98-C-4032, the EU contract IST-20001-33060, and the Danish Technical Research Council contract 26-01-0088.

References

Arkin, R. (1998). *Behaviour-Based Robotics.* MIT Press.

Beckers, R., Holland, O., and Deneubourg, J. (1994). From action to global task: Stigmergy and collective robotics. In *Proceedings of Artificial Life 4*, pages 181–189, Cambridge, Massachusetts, USA.

Beckers, R., Holland, O., and Deneubourg, J.-L. (2000). From local actions to global tasks: Stigmergy and collective robotics. *Prerational intelligence: Adaptive behavior and intelligent systems without symbols and logic*, 2:549–563.

Bojinov, H., Casal, A., and Hogg, T. (2000a). Emergent structures in modular self-reconfigurable robots. In *Proceedings of the IEEE int. conf. on Robotics & Automation*, volume 2, pages 1734 –1741, San Francisco, California, USA.

Bojinov, H., Casal, A., and Hogg, T. (2000b). Multiagent control of self-reconfigurable robots. In *Proceedings of the Fourth int. conf. on MultiAgent Systems*, pages 143 –150, Boston, Massachusetts, USA.

Butler, Z., Kotay, K., Rus, D., and Tomita, K. (2001). Cellular automata for decentralized control of self-reconfigurable robots. In *ICRA 2001 Workshop on Modular Self-Reconfigurable Robots*, Seoul, Korea.

Castano, A., Chokkalingam, R., and Will, P. (2000a). Autonomous and self-sufficient conro modules for reconfigurable robots. In *Proceedings of the 5th int. Symposion on Distributed Autonomous Robotic Systems*, pages 155–164, Knoxville, Texas, USA.

Castano, A., Shen, W.-M., and Will, P. (2000b). Conro: Towards deployable robots with inter-robot metamorphic capabilities. *Autonomous Robots*, 8(3):309–324.

Dekhil, M., Sobh, T., and Efros, A. (1996). Commanding sensors and controlling indoor autonomous mobile robots. In *Proceedings of the 1996 IEEE International Conference on Control Applications*, pages 199 –204, Dearborn, Michigan, USA.

Fukuda, T. and Nakagawa, S. (1990). Method of autonomous approach, docking and detaching between cells for dynamically reconfigurable robotic system cebot. *JSME int. Journal*, 33(2):263–268.

Hardy, N. and Ahmad, A. (1999). De-coupling for reuse in design and implementation using virtual sensors. *Autonomous Robots*, 6:265–280.

Henderson, T. and Shilcrat, E. (1984). Logical sensor systems. *Robotic Systems*, 1(2):169–193.

Hosokawa, K., Tsujimori, T., Fujii, T., Kaetsu, H., Asama, H., Kuroda, Y., and Endo, I. (1998). Self-organizing collective robots with morphogenesis in a vertical plane. In *Proceedings of the IEEE int. conf. on Robotics & Automation*, pages 2858–2863, Leuven, Belgium.

Intanagonwiwat, C., Govindan, R., and Estrin, D. (2000). Directed diffusion: A scalable and robust communication paradigm for sensor networks. In *Proceedings of the Sixth Annual International Conference on Mobile Computing and Networking (MobiCOM '00)*, pages 56–67, Boston, Massachussetts, USA.

Khoshnevis, B., Kovac, B., Shen, W.-M., and Will, P. (2001). Reconnectable joints for self-reconfigurable robots. In *Proceedings of the IEEE/RSJ int. conf. on Intelligent Robots and Systems*, Maui, Hawaii, USA.

Kotay, K., Rus, D., Vona, M., and McGray, C. (1998). The self-reconfiguring robotic molecule. In *Proceedings of the IEEE int. conf. on Robotics & Automation*, pages 424–431, Leuven, Belgium.

Matarić, M. J. (1997). Behavior-based control: Examples from navigation, learning, and group behavior. *Journal of Experimental and Theoretical Artificial Intelligence*, 9(2–3):323–336.

Murata, S., Kurokawa, H., and Kokaji, S. (1994). Self-assembling machine. In *Proceedings of the IEEE int. conf. on Robotics & Automation*, pages 441–448, San Diego, USA.

Murata, S., Kurokawa, H., Yoshida, E., Tomita, K., and Kokaji, S. (1998). A 3-d self-reconfigurable structure. In *Proceedings of the IEEE int. conf. on Robotics & Automation*, pages 432–439, Leuven, Belgium.

Murata, S., Yoshida, E., Tomita, K., Kurokawa, H., Kamimura, A., and Kokaji, S. (2000). Hardware design of modular robotic system. In *Proceedings of the IEEE/RSJ int. conf. on Intelligent Robots and Systems*, pages 2210–2217, Takamatsu, Japan.

Pamecha, A., Chiang, C., Stein, D., and Chirikjian, G. (1996). Design and implementation of metamorphic robots. In *Proceedings of the ASME Design Engineering Technical conf. and Computers in Engineering conf.*, pages 1–10, Irvine, USA.

Rowland, J. and Nicholls, H. (1989). A modular approach to sensor integration in robotic assembly. In Puente, E. and Nemes, L., (Eds.), *Information control problems in Manufacturing Technology*, pages 371–376. IFAC, Pergamon Press, Oxford, UK.

Rus, D. and Vona, M. (2000). A physical implementation of the crystalline robot. In *Proceedings of the IEEE int. conf. on Robotics & Automation*, pages 1726–1733, San Francisco, USA.

Rus, D. and Vona, M. (2001). Crystalline robots: Self-reconfiguration with compressible unit modules. *Autonomous Robots*, 10(1):107–124.

Salemi, B., Shen, W., and Will, P. (2001). Hormone controlled metamorphic robots. In *Proceedings of the IEEE int. conf. on Robotics & Automation*, pages 4194–4199, Seoul, Korea.

Shen, W.-M., Salemi, B., and Will, P. (2000a). Hormone-based control for self-reconfigurable robots. In *Proceedings of the int. conf. on Autonomous Agents*, pages 1–8, Barcelona, Spain.

Shen, W.-M., Salemi, B., and Will, P. (2000b). Hormones for self-reconfigurable robots. In *Proceedings of the int. conf. on Intelligent Autonomous Systems*, pages 918–925, Venice, Italy.

Støy, K. (2001). Using situated communication in distributed autonomous mobile robots. In *Proceedings of the 7th Scandinavian conf. on Artificial Intelligence*, Odense, Denmark.

Støy, K., Shen, W.-M., and Will, P. (2002a). Global locomotion from local interaction in self-reconfigurable robots. In *Proceedings of the 7th int. conf. on Intelligent Autonomous Systems IAS-7 (to appear)*, Marina del Rey, California, USA.

Støy, K., Shen, W.-M., and Will, P. (2002b). How to make a self-reconfigurable robot run. In *Proceedings of the First International Joint Conference on Autonomous Agents & Multiagent Systems (to appear)*, Bologna, Italy.

Ünsal, C. and Khosla, P. (2000). Mechatronic design of a modular self-reconfiguring robotic system. In *Proceedings of the IEEE int. conf. on Robotics & Automation*, pages 1742–1747, San Francisco, USA.

Yim, M. (1994). New locomotion gaits. In *Proceedings of int. conf. on Robotics & Automation*, pages 2508 –2514, San Diego, California, USA.

Yim, M., Duff, D., and Roufas, K. (2000). Polybot: A modular reconfigurable robot. In *Proceedings of the IEEE int. conf. on Robotics & Automation*, pages 514–520, San Francisco, USA.

How to Make a Self-Reconfigurable Robot Run

K. Støy
The Maersk Mc-Kinney Moller Institute for Production Technology
Campusvej 55
DK-5230 Odense M, Denmark
kaspers@mip.sdu.dk

W.-M. Shen
Information Sciences Institute and Computer Science Department
4676 Admiralty way
Marina del Rey, CA 90292, USA
shen@isi.edu

P. Will
Information Sciences Institute and Computer Science Department
4676 Admiralty way
Marina del Rey, CA 90292, USA
will@isi.edu

ABSTRACT

In this paper we present role based control which is a multi-agent based control algorithm for self-reconfigurable robots. We use role based control to implement quadruped and hexapod gaits in a real self-reconfigurable robot made from up to nine independent autonomous modules. We show that this implementation scales and argue that it is minimal, robust to module failures, to loss of communication signals, and to interchange of modules.

In role based control all modules of the robot run identical programs, but may play different *roles*. The modules decide what role to play based on their local configuration and information propagated down to them through the configuration tree. A role consists of a cyclic motion, the period of this motion, and a set of delays. The delays specify the phase delay of the cyclic motions of the child modules compared to the parent. These delays are used to coordinate the motions of the individual module to obtain a coordinated global behavior.

Categories and Subject Descriptors

I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence—*Multiagent systems, coherence and coordination*; I.2.9 [**Artificial Intelligence**]: Robotics—*Autonomous vehicles*

General Terms

Algorithms, Experimentation

1. INTRODUCTION

Reconfigurable robots are robots made from a possibly large number of independent modules that are connected to form a robot. If the modules from which the reconfigurable robot is built are able to connect and disconnect without human intervention the robot is a self-reconfigurable robot.

AAMAS'02 July 15-19, 2002, Bologna, Italy.

An example of a self-reconfigurable robot is the CONRO robot. A module of the CONRO system is shown in Figure 1. Examples of other physically realized self-reconfigurable robots can be found in [4, 6, 7, 8, 9, 10, 11, 13, 17, 19, 20].

Several potential advantages of self-reconfigurable robots over traditional robots have been pointed out in literature:

- **Versatility**. The modules can be combined in different ways making the same robotic system able to perform a wide range of tasks [8, 15].
- **Adaptability**. While the self-reconfigurable robot performs its task it can change its physical shape to adapt to changes in the environment [13].
- **Robustness**. Self-reconfigurable robots are made from many identical modules and therefore if a module fails it can be replaced by another [7, 19, 15].
- **Cheap production**. When the final design for the basic module has been obtained it can be mass produced and thereby keep the cost of the individual module low compared to its complexity. [7, 8, 19].

Self-reconfigurable robots can solve the same tasks as traditional robots, but as Yim et al [19] point out, in applications where the task and environment are given a priori it is often cheaper to build a special purpose robot. Therefore, the applications best suited for self-reconfigurable robots are applications where some leverage can be gained from the special abilities of self-reconfigurable robots.

The versatility of these robots make them suitable in scenarios where the robots have to handle a range of tasks. The robots can also handle tasks in unknown or dynamic environments, because they are able to adapt to these environment. In tasks where robustness is of importance it might be desirable to use self-reconfigurable robots. Even though real applications for self-reconfigurable robots still are to be seen, a number of specific applications have been envisioned [13, 8, 19]: fire fighting, search and rescue after an earthquake, battlefield reconnaissance, planetary exploration, undersea mining, and space structure building. Other possible applications include entertainment and service robotics.

The potential of self-reconfigurable robots can be realized if several challenges in terms of hardware and software can be met. In this work we focus on one of the challenges in software: how do we make a large number of connected modules perform a coordinated global behavior? Specifically we

address how to design algorithms that will make it possible for self-reconfigurable robots to locomote efficiently. In order for a locomotion algorithm to be useful it has to preserve the special properties of these robots. From the advantages and applications mentioned above we can extract a number of guidelines for the design of such a control algorithm. The algorithm should be distributed to avoid having a single point of failure. The performance of the algorithm should scale with an increased number of modules. It has to be robust to reconfiguration, because reconfiguration is a fundamental capability of self-reconfigurable robots. Finally, it is desirable to have homogeneous software running on all the modules, because it makes it possible for any module to take over if another one fails.

It is an open question if a top-down or a bottom-up approach gives the best result. We find that it is difficult to design the system at a global level and then later try to distribute it, because often properties of the hardware are ignored and a slow robotic system might be the result. Therefore, we use a bottom-up approach where the single module is the basic unit of design. That is, we move from a global design perspective to a bottom-up one where the important design element is the individual module and its interactions with its neighbors. The global behavior of the system then emerges from the local interaction between individual modules. This way each module plays the role of an agent in a multiagent system. A similar approach is also used by Bojinov et al [1, 2].

2. RELATED WORK

In the related work presented here we focus on control algorithms for locomotion of self-reconfigurable robots.

Yim et al [18, 19] demonstrate among other types of locomotion a four-legged spider like type of locomotion. In their system each module has a gait control table where each column represents the actions performed by one module. Motion is then obtain by having a master synchronizing the transition from one row to the next. The problem with this approach is the need for a central controller, since it gives the system a single point of failure. If there is no master it is suggested that the modules can be assumed to be synchronized in time and each module can execute its column of actions open-loop. However, since all the modules are autonomous it is a questionable assumption to assume that all the modules are and can stay synchronized. In order to use the gait control table each module needs to know what column it has to execute. This means that the modules need IDs. Furthermore, if the configuration changes or the number of modules changes the gait control table has to be rewritten.

Shen and Salemi propose to use artificial hormones to synchronize the modules to achieve consistent global locomotion. In earlier versions of the system a hormone is propagated through the self-reconfigurable system to achieve synchronization [13]. In later work the hormone is also propagated backwards making all modules synchronized before a new action is initiated [14, 12]. This synchronization takes time $O(n)$ where n is the number of modules. This slows down the system considerably, because it has to be done before each action. Also, the entire system stops working if one hormone is lost. This can easily happen due to unreliable communication, a module disconnecting itself before a response can be given, or a module failure. In fact, the system has n-points of failure which is not desirable. The earlier version is better in this sense, but still performance remains low because a synchronization hormone is sent before each action.

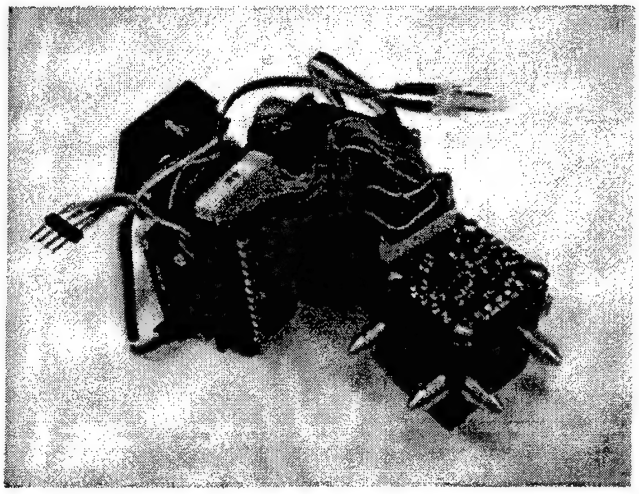

Figure 1: A CONRO module.

In our system all modules repeatedly go through a cyclic sequence of joint angles describing a motion. This sequence could come from a column in a gait control table, but in our implementation the joint angles are calculated using a cyclic function. Every time a module reaches a specified position p in the cycle a message is sent through a specified child connector. If the signal is received the child module resets its position in its cycle making it delayed p compared to the parent. This way the actions of the individual module are decoupled from the synchronization mechanism resulting in a faster and more reliable system. Furthermore, there is no need to make changes to the algorithm if the number of modules changes.

3. ROLE BASED CONTROL

In our approach we acknowledge the need for each module to be autonomous in order to obtain a robust and scalable system. We also acknowledge the need for a tight coupling between the modules to coordinate and produce the desired global behavior. However it is not desirable to coordinate at the level of the individual motor control command, because involving other modules in low level control will produce a less responsive system. Therefore, we abstract away the low level control and coordinate at a higher level. We coordinate at the level of *roles*. In the following we will define what a role is. We will describe the algorithm a module uses to play a role, and finally we will discus how modules can decide to change roles over time.

3.1 A Role

A role consists of three components. The first component is a function $A(t)$ that specifies the joint angles of a module given an integer $t \in [0 : T]$, where T is the period of the motion and the second component that needs to be specified. The third component is a set of delays D. A delay $d_i \in D$ specifies the delay between the child connected to connector i and the parent. That is, if the parent is at step $t_{parent} = t_1$ the child is at $t_{child} = (t_1 - d_i + T)$ *modulus* T.

```
t = 0
while(1)
  if (t=d_1) then <send signal to child connector 1>
  ...
  if (t=d_n) then <send signal to child connector n>

  if <signal received from parent connector> then
    t=0
  endif

  <perform action A(t)>
  t = (t+1) modulus T
endwhile
```

Figure 2: The algorithm used to play a role. Refer to section 3.2 for further explanation.

3.2 Playing a Role

The algorithm that we are now about to describe is used to make a module play a role. However first some assumptions need to be made: a parent connector is specified and the remaining connectors are considered child connectors, connections can only be made between a parent connector and a child connector. These assumptions limit the configuration the algorithm can handle to tree configurations. Under these assumptions a role is played using the algorithm shown in Figure 2.

Ignoring the if-statements in the beginning of the loop, the module repeatedly goes through a sequence of actions parameterized by t. This part of the algorithm alone makes a single module repeatedly perform the sequence of actions specified by $A(t)$. However, in order to achieve coherent global behavior the module needs to be synchronized with neighbors. Therefore, at $t_{parent} = d_i$ a signal is sent through child connector i. Note that it does not matter to the parent if a child module actually receives the signal, because in that case the signal will just be lost. However, if child i receives the signal it sets $t_{child-i} = 0$. This enforces that the child is delayed d_i compared to the parent.

A simple rule that will come in handy when we calculate the delays for a specific locomotion pattern is that we can calculate what step a child $t_{child-i}$ is at based on what step the parent t_{parent} is at:

$$t_{parent} = t_1 \quad \Rightarrow \quad t_{child-i} = t_1 - d_i \tag{1}$$

The other way:

$$t_{child} = t_1 \quad \Rightarrow \quad t_{parent} = t_1 + d_i \tag{2}$$

In terms of execution time we can see that from the time the modules are connected, it takes time proportional to the height of the tree for all the modules to synchronize. However, once the modules are synchronized, the algorithm keeps the modules synchronized using only constant time. Below is an example of a caterpillar role.

$$A(t) = \begin{cases} pitch(t) = 50^\circ \sin(\frac{2\pi}{T}t) \\ yaw(t) = 0 \end{cases} \tag{3}$$

$$d_{north} = \frac{T}{5} \tag{4}$$

$$T = 180 \tag{5}$$

```
r = <start role>
t = 0
while(1)
  if (t=d(r)_1) then
    <send message M(r,1) to child connector 1>
    <update r>
 endif
  ...
  if (t=d(r)_n) then
    <send message M(r,n) to child connector n>
    <update r>
  endif

  if <message m received from parent connector> then
    t=0
    <update r based on m>
  endif

  <perform action A(r,t)>
  t = (t+1) modulus T(r)
endwhile
```

Figure 3: The algorithm used to enable modules to play different roles depending on their position in the configuration tree. Refer to section 3.3 for further explanation.

In earlier work we have demonstrated that when the modules are connected in a chain and all play this role they produce caterpillar like locomotion [15]. In this work we also demonstrated a locomotion pattern similar to that of a sidewinder snake.

3.3 Combining Roles

In simple locomotion gaits only one role is needed to obtain the desired global behavior. However, in more complex locomotion patterns more roles will be needed. For instance, in a walking robot the following roles can be identified: left leg, right leg, and spine. In order to handle these more complex locomotion patterns we need to extend our algorithm.

It is obvious that we have to define each of the roles needed to produce the desired global behavior. For each role r we supply an action sequence $A(r,t)$, a period $T(r)$, and a set of delays $D(r)$. We also need to define how a module decides when to change role.

A module can change its role in two ways. One way is as a reaction to changes in the local configuration. The other way is in response to a message from the parent. We encode the information needed to make these decisions into the communication signals that are already part of the synchronization mechanism. It does not make sense to implement it as a separate communication mechanism for two reasons. 1) It is of no value to the module to know what role to play before it is synchronized. 2) It adds more complexity to the system. The resulting algorithm is shown in Figure 3.

The algorithm looks very similar to the basic algorithm. The main difference is that now the synchronization signal contains a message that is a function $M(r,i)$ of the role r the module is playing and the connector i the message is sent through. This function can be used to uniquely specify what role each module in the system should play. The other difference is that based on the success of communication the module can detect its local configuration and use this information to update its role appropriately.

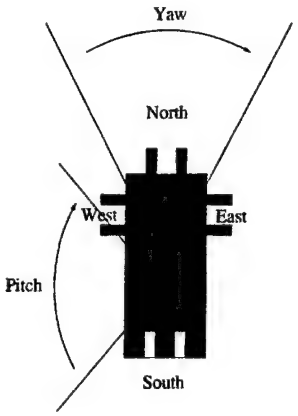

Figure 4: A schematic overview of the CONRO module. The connectors are labeled with compass directions. The arrows indicate the direction of increasing angle.

4. THE CONRO MODULES

For our experiments we use a self-reconfigurable robot made from the CONRO modules developed at University of Southern California's Information Sciences Institute [3, 5] (see figure 1). The modules are roughly shaped as rectangular boxes measuring 10cm x 4.5cm x 4.5cm and weigh 100grams. The modules have a female connector located at one end facing south and three male connectors located at the other end facing east, west, and north (see Figure 4). Each connector has an infra-red transmitter and receiver used for local communication and sensing. The modules have two controllable degrees of freedom: pitch (up and down) and yaw (side to side). Processing is taken care of by an onboard Basic Stamp 2 processor. The modules have onboard batteries, but these do not supply enough power for the experiments reported here and therefore the modules are powered through cables. Refer to http://www.isi.edu/conro for more details and for videos of the experiments reported later in this paper.

5. IMPLEMENTING A WALKING GAIT

Our goal is to implement a walking gait in the CONRO self-reconfigurable robot configured as shown in Figure 6. In order to do so we need to define three different roles: spine, east leg, and west leg. The two main tasks are to specify the action sequences and the delays for each role. We will look at these two problems below.

5.1 Actions

We start out by specifying the actions for each role. Intuitively the legs should be lifted from the ground when moving forward and touching the ground when moving backwards. We use the following motion equation for the east legs:

$$A(east\,leg,t) = \begin{cases} pitch(t) = 35^\circ \cos(\frac{2\pi}{T}t) - 55^\circ \\ yaw(t) = 40^\circ \sin(\frac{2\pi}{T}t) \end{cases} \quad (6)$$

The equation for the west legs is obtained by replacing t by $2\pi - t$ giving the same motion, but in the opposite direction. This motion is visualized in Figure 5.

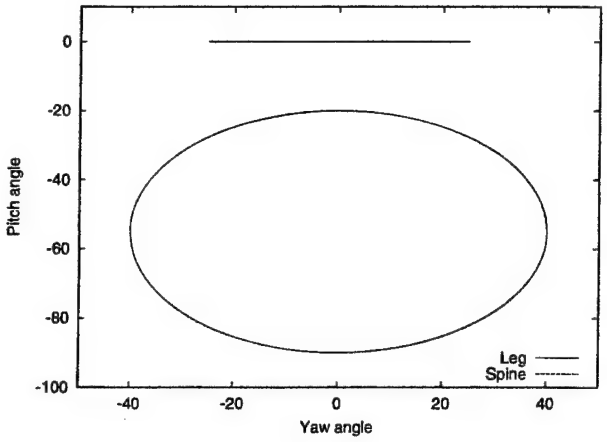

Figure 5: The motion of a module playing the leg role (top) and the spine role (bottom) visualized in joint space.

The spine module between two pairs of legs should bend from side to side to increase the length of each step. The parameters for this motion are shown below and are also visualized in Figure 5.

$$A(spine,t) = \begin{cases} pitch(t) = 0^\circ \\ yaw(t) = 25^\circ \cos(\frac{2\pi}{T}t + \pi) \end{cases} \quad (7)$$

For simplicity we pick the same period T for all roles. The parameter T can later be used to control the locomotion speed.

5.2 Delays

What is left is to coordinate the motion of the modules. The question is how do we get from a general description of which modules should be coordinated to the delays needed in our algorithm. In a walking robot the left front leg and the rear right should be synchronized. The same goes for the right front leg and the rear left leg. Also, it would result in a more efficient locomotion pattern if the spine modules bend to increase the length of each step.

We configure our modules as shown in Figure 6. We first consider the spine module located in the top middle part of the figure labeled spine-1. The question is now what fraction of a period each child should be delayed. We first turn our attention to the front legs: east-1 and west-1. It is obvious that the motion of these legs should be half a period apart. This way one leg will touch the ground when the other is lifted and the other way around. Therefore, the delay between the two legs should be $T/2$ ($+nT$ where $n \in [0:\infty]$ which we consistently omit from these calculations). Given t_{east-1} and t_{west-1} we use equation 2 to calculate what t this corresponds to in the spine-1 module.

$$\begin{aligned} t_{spine-1} &= t_{east-1} + d_{east} \\ t_{spine-1} &= t_{west-1} + d_{west} \end{aligned} \quad (8)$$

Setting the two expressions for $t_{spine-1}$ equal to each other we get:

$$t_{east-1} + d_{east} = t_{west-1} + d_{west} \quad (9)$$

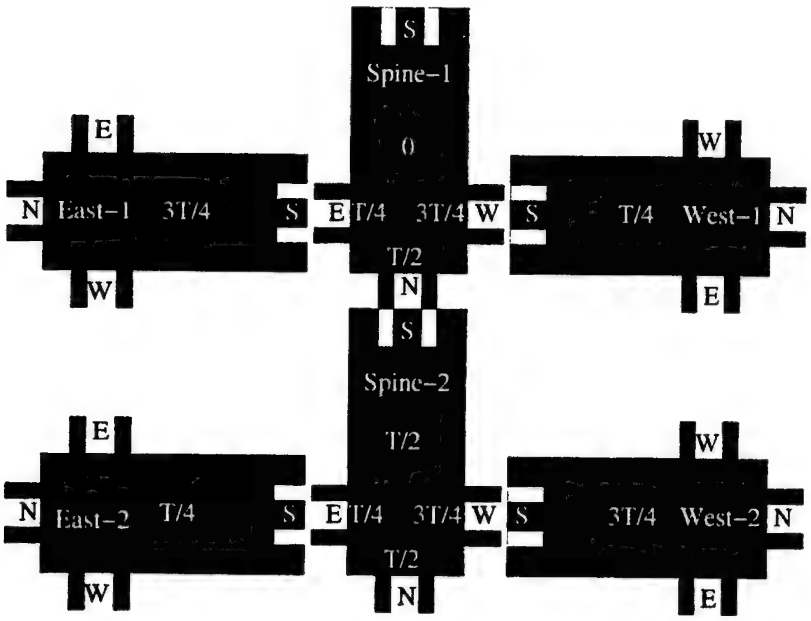

Figure 6: The black boxes represent modules. The modules are connected to form a quadruped robot. All the modules are named and labeled with compass directions. The expression next to a connector represents the delay d across that connector. The delay is expressed in fractions of a period T. For instance, the delay associated with the east connector of the spine module is $d_{east} = T/4$. The expressions located in the center of the modules represent the value of t of that module when the spine module spine-1 is at $t_{spine-1} = 0$. The t value of child is calculated using $t_{child-i} = t_{parent} - d_i$ where i is the connector to which the child is connected. Note that by using the delays shown here the top east leg and bottom west legs are synchronized. This also goes for the top west and bottom east leg.

We exploit that we want the difference between t_{east-1} and t_{west-1} to be $T/2$ to find the following constraint:

$$d_{east} - d_{west} = \frac{T}{2} \quad (10)$$

A similar consideration leads to the conclusion that the front west leg and the rear east leg should be synchronized. However the signal to the rear east leg is delayed when it goes though the north connector as well. Given t_{east-2} and t_{west-1} we again using rule 2 to transform into $t_{spine-1}$.

$$t_{spine-1} = t_{west-1} + d_{west} \quad (11)$$

$$t_{spine-1} = t_{east-2} + d_{east} + d_{north} \quad (12)$$

Setting the two expressions for $t_{spine-1}$ equal to each other we get:

$$t_{west-1} + d_{west} = t_{east-2} + d_{east} + d_{north} \quad (13)$$

We exploit that we want $t_{west-1} - t_{east-2} = 0$ and equation 10 to obtain:

$$d_{north} = \frac{T}{2} \quad (14)$$

Using our knowledge about how the legs should be synchronized we have come up with constraints for the delays. We know that a module playing the spine role receives a message from the parent at $t = 0$ and has to send one through the north connector at $t = T/2$. In order to spread out the communication over a period we pick the following values for d_{east} and d_{west} that satisfies constraint 10.

$$d_{east} = \frac{T}{4} \quad (15)$$

$$d_{west} = \frac{3T}{4} \quad (16)$$

Now all the delays are specified, but there is one piece missing. We have to make sure that the spine also is coordinated. We know from the spine motion defined in equation 7 that the spine is bent most to the east at $t_{spine-2} = T/2$. The east legs are closest together at $t_{east-2} = T/4$ (see equation 6). We want the difference between $t_{spine-2}$ and t_{east-2} to be zero. Again applying equation 2 we transform t_{east-2} into $t_{spine-2}$:

$$t_{spine-2} = \frac{T}{4} + t_{east-2} \quad (17)$$

$$t_{spine-2} = T/2 \quad (18)$$

Setting the two equations equal to each other we in fact see that our choice for t_{east} makes sure that the spine is indeed making the robot take longer steps. These results are summarized in Figure 6.

We have now calculated the delays for the spine module. If feet-modules were connected to the legs we would have to calculate delays for the legs as well. However, we are not planning to connect any modules so for simplicity we just give the leg modules the same delays as the spine modules.

5.3 Role selection

Now all the modules are synchronized and can play the role of a spine, a east leg, or west leg. The question is now how each module decides what part it plays. We have to define the function $M(r, i)$ that maps a role r and a connector i to a message. In our simple configuration we define M as:

$$M(r,i) = i, \quad where\, i \in \{south, east, west\} \tag{19}$$

This intuitively means that a message contains information about which connector it was sent through. Upon receiving a message m, the role selection is straight forward:

$$r(m) = \begin{cases} west\, leg, & if \quad m = west \\ east\, leg, & if \quad m = cast \\ spine, & if \quad m = south \end{cases} \tag{20}$$

These role selection rules correctly assign the roles to modules, even if the modules are interchanged. There is one notable exception. The root of the tree never receives any messages, because it by definition does not have a parent. Therefore we need to introduce rules that based on the local configuration can detect if a module is the root and therefore should play the spine role. In the quadruped configuration this is easy. We simply say that if the module successfully communicate with a child connected either to the west or east connector, the module changes its role to a spine role (if it is not already a spine module).

The role selection rules are arbitrary since many role selection rules and functions M exist that would lead to the same behavior. In systems where there are more roles more effort has to be put into defining rules to make sure the rules are not ambiguous.

6. EXPERIMENTS

In general, it is problematic to report performance of a self-reconfigurable system, because there is such a tight coupling between hardware and software. In this work we report scalability of the algorithm. Furthermore we report the length of our programs as a measure of the complexity of the control algorithm. We also report the speed of the walking gaits, but this should only be considered an example, the reason being that in our system the limiting factors are how robust the modules physically are, how powerful the motors are, and how much power we can pull from the power source. To report a top speed is not meaningful before the robot runs autonomously on batteries.

6.1 Quadruped Locomotion

In the first experiment we assembled the modules in the quadruped configuration shown in Figure 7. We then measured the time it took for the robot to walk a distance of 150cm. We found that the average of ten trials was 10.9seconds and the standard deviation was 0.57seconds. This corresponds to a speed of 13.8cm/second.

The main loop of the program excluding comments and labels takes up 120 lines of code. The initialization part contains 32 lines of code. The small size of the program emphasizes the point that the control algorithm is simple and minimal. In this system we can replace modules or move them around as desired, because they will pick the right role and synchronize correctly no matter where they are placed in the configuration.

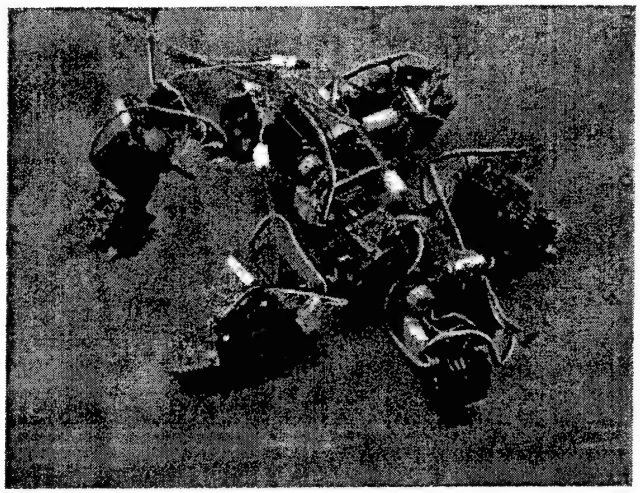

Figure 7: The robot performing quadruped locomotion. The wires connected to each module only provide power.

6.2 Hexapod Locomotion

We extended the quadruped with an extra pair of leg and a spine module to obtain a hexapod robot. This configuration can be seen in Figure 8. Note that the controllers of the module do not need to be changed, because the delays make sure that the third pair of legs is appropriately delayed. We repeated the experiments another ten times and found that the average time was 12.0seconds (12.5cm/sec.) and the standard deviation was 0.57seconds.

Initially we tested the hypothesis that the speed of the robot is independent of the number of modules. Unfortunately Student's t-test rejected this hypothesis (test probability $3.0 * 10^{-7}$). From close observation of the experiments we found that the quadruped robot makes longer step, because it slides a little forward with each step due to its momentum. In the hexapod this is not the case, because of the friction caused by the extra pair of legs. In order to remove this difference from our data we returned to the videos of the experiments and counted the number of steps taken by the robot in each experiment. We divided the time with the number of steps to produce a time per step measure. We then tested the hypothesis that the time per step is the same for both the quadruped and hexapod walker. This hypothesis was accepted on the 5% confidence level with test probability 0.78. This implies that the speed of the system does not dependent on the number of modules and therefore the algorithm scales. If we had more modules available we could extend the robot to make a 2n-legged walker. The algorithm can handle this because after the initial synchronization it only takes constant time per period to keep the modules synchronized.

7. DISCUSSION

The system achieves a high level of performance because we use a less aggressive synchronization mechanism. Intuitively the idea is that as long as each module keeps its children synchronized each period then all the modules will become and stay synchronized over time. This idea only works if it can be assumed that one period of motion takes

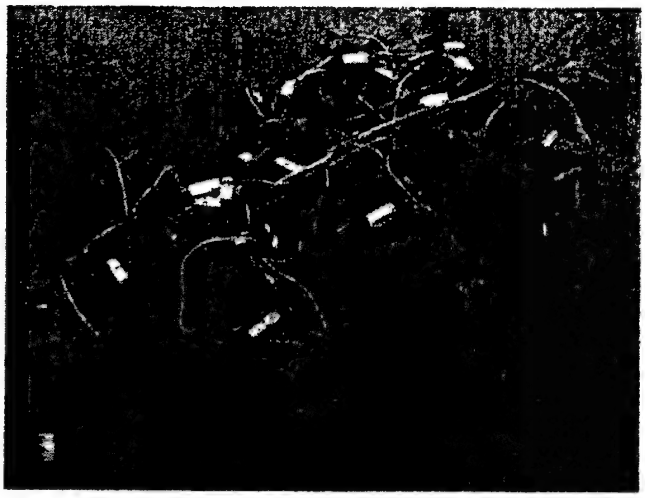

Figure 8: The robot performing hexapod locomotion. The wires connected to each module only provide power.

the same amount of time for all modules otherwise the modules will keep getting out of synchronization. To insure that the periods take the same amount of time is not a problem in our simple system, but as the system grows more complex and the module's resources are needed for other tasks it will be necessary to put some extra work into making the timing of each period precise enough. Fortunately, this can be achieved by using timers or interrupts.

When the modules are synchronized they can stay synchronized for some cycles without communicating, because the time to complete a cycle is approximately the same for all modules. This means that it does not matter much if a synchronization signal is lost as long as one makes it through from time to time.

In the algorithm the root module of the configuration tree emerges as the leader of the robot. This does not mean that there is a single point of failure, because if the root module fails its children will take over. In this situation each child will become the root of its own sub-tree. However, it can of course not be guaranteed that these sub-trees remain synchronized. This simple implicit leader selection mechanism is very powerful, but unfortunately it doesn't work if the configuration contains loops. In a loop synchronization signals can chase each other around without ever reaching each other. In [15] we have solved this problem by introducing IDs and combining a simple leader selection algorithm with the basic role playing algorithm.

Another point to note is that the action sequences are just open loop motor control commands. This is not desirable if the robot is to operate in complex environments where sensor feedback is essential to the survival of the robot. The method can be extend to include this form of feedback. For instance, the cyclic motion of a leg can be biased by feedback from the environment. This way the legs can change the motion to avoid obstacles or gaps. Including sensor feedback at the level of the individual module will only make the robot able to deal with problems that can be solved at the level of the individual module. If the robot has to avoid an obstacle it requires coordinated actions of all the modules to avoid the obstacle. A basic approach we have investigated in [16] is to let all the modules share sensor information through propagation. Each module then changes its role locally to react to this sensor information.

Finally another interesting question is: can this algorithm be generalized to a system without a parent-child relationship. In a real environment a single module should be able to influence the control of the entire robot based on some critical sensor information only that module has access to. How to do that is the focus of our future research.

8. SUMMARY

We have introduced a general multiagent based algorithm that can be used to implement locomotion patterns in a self-reconfigurable robot. In this algorithm each module plays a *role*. The *role* can be changed either by communication from a parent module or by detecting changes in the local configuration. It has been described how modules playing roles are synchronized. We have used this general algorithm to implement a walking gait in a self-reconfigurable robot consisting of up to 9 modules. We show in experiments that the implemented algorithm scales and is an efficient implementation of both a quadruped and hexapod gait.

9. ACKNOWLEDGMENTS

This research is supported by the DARPA contract DAAN 02-98-C-4032, the AFOSR contract F49620-01-1-0020, the EU-contract IST-20001-33060: HYDRA "Living" Building Blocks for Self-Designing Artifacts, and the Danish Technical Research Council contract 26-01-0088.

10. REFERENCES

[1] H. Bojinov, A. Casal, and T. Hogg. Emergent structures in modular self-reconfigurable robots. In *Proceedings of the IEEE International Conference on Robotics & Automation*, volume 2, pages 1734 –1741, San Francisco, California, USA, 2000.

[2] H. Bojinov, A. Casal, and T. Hogg. Multiagent control of self-reconfigurable robots. In *Proceedings of the Fourth International Conference on MultiAgent Systems*, pages 143 –150, Boston, Massachusetts, USA, 2000.

[3] A. Castano, R. Chokkalingam, and P. Will. Autonomous and self-sufficient conro modules for reconfigurable robots. In *Proceedings of the 5th International Symposium on Distributed Autonomous Robotic Systems*, pages 155–164, Knoxville, Texas, USA, 2000.

[4] K. Hosokawa, T. Tsujimori, T. Fujii, H. Kaetsu, H. Asama, Y. Kuroda, and I. Endo. Self-organizing collective robots with morphogenesis in a vertical plane. In *Proceedings of the IEEE International Conference on Robotics & Automation*, pages 2858–2863, Leuven, Belgium, 1998.

[5] B. Khoshnevis, B. Kovac, W.-M. Shen, and P. Will. Reconnectable joints for self-reconfigurable robots. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, Maui, Hawaii, USA, 2001.

[6] K. Kotay, D. Rus, M. Vona, and C. McGray. The self-reconfiguring robotic molecule. In *Proceedings of the IEEE International Conference on Robotics & Automation*, pages 424–431, Leuven, Belgium, 1998.

[7] S. Murata, H. Kurokawa, and S. Kokaji. Self-assembling machine. In *Proceedings of the IEEE International Conference on Robotics & Automation*, pages 441–448, San Diego, USA, 1994.

[8] S. Murata, H. Kurokawa, E. Yoshida, K. Tomita, and S. Kokaji. A 3-d self-reconfigurable structure. In *Proceedings of the IEEE International Conference on Robotics & Automation*, pages 432–439, Leuven, Belgium, 1998.

[9] S. Murata, E. Yoshida, K. Tomita, H. Kurokawa, A. Kamimura, and S. Kokaji. Hardware design of modular robotic system. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2210–2217, Takamatsu, Japan, 2000.

[10] A. Pamecha, C. Chiang, D. Stein, and G. Chirikjian. Design and implementation of metamorphic robots. In *Proceedings of the ASME Design Engineering Technical Conference and Computers in Engineering Conference*, pages 1–10, Irvine, USA, 1996.

[11] D. Rus and M. Vona. A physical implementation of the crystalline robot. In *Proceedings of the IEEE International Conference on Robotics & Automation*, pages 1726–1733, San Francisco, USA, 2000.

[12] B. Salemi, W. Shen, and P. Will. Hormone controlled metamorphic robots. In *Proceedings of the IEEE International Conference on Robotics & Automation*, pages 4194–4199, Seoul, Korea, 2001.

[13] W.-M. Shen, B. Salemi, and P. Will. Hormone-based control for self-reconfigurable robots. In *Proceedings of the International Conference on Autonomous Agents*, pages 1–8, Barcelona, Spain, 2000.

[14] W.-M. Shen, B. Salemi, and P. Will. Hormones for self-reconfigurable robots. In *Proceedings of the International Conference on Intelligent Autonomous Systems*, pages 918–925, Venice, Italy, 2000.

[15] K. Støy, W.-M. Shen, and P. Will. Global locomotion from local interaction in self-reconfigurable robots. In *Proceedings of the 7th International Conference on Intelligent Autonomous Systems IAS-7*, Marina del Rey, California, USA, 2002.

[16] K. Støy, W.-M. Shen, and P. Will. On the use of sensors in self-reconfigurable robots. In *Proceedings of the Seventh International Conference on The Simulation of Adaptive behavior SAB'02 (to appear)*, Edinburgh, UK, 2002.

[17] C. Ünsal and P. Khosla. Mechatronic design of a modular self-reconfiguring robotic system. In *Proceedings of the IEEE International Conference on Robotics & Automation*, pages 1742–1747, San Francisco, USA, 2000.

[18] M. Yim. *Locomotion with a unit-modular reconfigurable robot.* PhD thesis, Department of Mechanical Engineering, Stanford University, 1994.

[19] M. Yim, D. Duff, and K. Roufas. Polybot: A modular reconfigurable robot. In *Proceedings of the IEEE International Conference on Robotics & Automation*, pages 514–520, San Francisco, USA, 2000.

[20] E. Yoshida, S. Murata, S. Kokaji, K. Tomita, and H. Kurokawa. Micro self-reconfigurable robotic system using shape memory alloy. In *Distributed Autonomous Robotic Systems 4*, pages 145–154, Knoxville, USA, 2000.

Global Locomotion from Local Interaction in Self-Reconfigurable Robots

K. Støy[1], W.-M. Shen[2] and P. Will[2]

kaspers@mip.sdu.dk {shen,will}@isi.edu

[1]The Maersk Mc-Kinney Moller Institute for Production Technology
University of Southern Denmark, Campusvej 55, DK-5230 Odense M, Denmark
[2]Information Sciences Institute, University of Southern California
4676 Admiralty Way, Marina del Rey, CA 90292, USA

Abstract. We present a general distributed control algorithm for achieving locomotion of a self-reconfigurable robot. In this algorithm each module continuously performs a cyclic sequence of actions with a period T. When a specified fraction of this period d has elapsed a signal is sent to all child modules. Upon receiving this signal the child module resets its action sequence making it delayed d compared to its parent. The algorithm is minimal and robust to loss of synchronization signals and change in the number of modules. We show in three different experiments that the algorithm can be used to implement a caterpillar, a sidewinder, and a rolling wheel gait in a real self-reconfigurable robot consisting of eight modules.

1 Introduction

Reconfigurable robots are robots made from a possibly large number of independent modules connected to form a robot. If the modules from which the reconfigurable robot is built are able to connect and disconnect without human intervention the robot is a self-reconfigurable robot. Examples of physically realized self-reconfigurable robots can be found in [8, 6, 9, 15, 13, 11, 7].

Several potential advantages of self-reconfigurable robots over traditional robots have been pointed out in literature:

- **Versatility**. The modules can be combined in different ways making the same robotic system able to perform a wide range of tasks.
- **Adaptability**. While the self-reconfigurable robot performs its task it can change its physical shape to adapt to changes in the environment.
- **Robustness**. Self-reconfigurable robots consist of many identical modules and therefore if a module breaks down it can be replaced by another.
- **Cheap production**. When the final design for the basic module has been obtained it can be mass produced and thereby keep the cost of the individual module low.

Self-reconfigurable robots can solve the same tasks as traditional robots, but as Yim et al [15] point out; in applications where the task and environment are given a priori it is

often cheaper to build a special purpose robot. Therefore, applications best suited for self-reconfigurable robots are applications where some leverage can be gained from the special abilities of self-reconfigurable robots. The versatility of these robots make them suitable in scenarios where the robots have to handle a range of tasks. The robots can also handle tasks in unknown or dynamic environments, because they are able to adapt to these environments. In tasks where robustness is of importance it might be desirable to use self-reconfigurable robots. Even though real applications for self-reconfigurable robots still are to be seen, a number of applications have been envisioned [11, 15]: fire fighting, search and rescue after an earthquake, battlefield reconnaissance, planetary exploration, undersea mining, and space structure building. Other possible applications include entertainment, service robotics, and payload management.

The potential of self-reconfigurable robots can be realized if several challenges in terms of hardware and software can be met. In this work we focus on one of the challenges in software: how do we make a large number of connected modules perform a coordinated global behavior? Specifically we address how to design algorithms that will make it possible for self-reconfigurable robots to locomote efficiently. In order for a locomotion algorithm to be useful it has to preserve the special properties of these robots. From the advantages and applications mentioned above we can extract a number of guidelines for the design of such a control algorithm. The algorithm should be distributed to avoid having a single point of failure. Also the performance of the algorithm should scale with an increased number of modules. It has to be robust to reconfiguration, because reconfiguration is a fundamental capability of self-reconfigurable robots. Finally, it is desirable to have homogeneous software running on all the modules, because it makes it possible for any module to take over if another one fails.

It is an open question if a top-down or a bottom-up approach gives the best result. We find that it is difficult to design the system at a global level and then later try to distribute it, because often properties of the hardware are ignored and a slow robotic system might be the result. Therefore, we use a bottom-up approach where the single module is the basic unit of design. That is, we move from a global design perspective to a bottom-up one where the important design element is the individual module and its interactions with its neighbors. The global behavior of the system then emerges from the local interaction between individual modules. A similar approach is also used by Bojinov et al [1].

2 Related Work

In the related work presented here we focus on control algorithms for locomotion of self-reconfigurable robots.

Yim et al [14, 15] demonstrate caterpillar like locomotion and a rolling track. Their system is controlled based on a gait control table . Each column in this table represents the actions performed by one module. Motion is then obtain by having a master synchronizing the transition from one row to the next. The problem with this approach is that the amount of communication needed between the master and the modules will limit its scalability. Another problem is the need for a central controller, since it gives the system a single point of failure. If there is no master it is suggested that the modules can be assumed to be synchronized in time and each module can execute its column of actions open-loop. However, since all the modules are autonomous it is a questionable assumption to assume that all the modules are and can stay synchronized. In order to use the gait control table each module needs to know what column it has to execute. This means that the modules need IDs. Furthermore, if the configuration changes or the number of modules changes the table has to be rewritten.

Shen, Salemi, and others propose to use artificial hormones to synchronize the modules to achieve consistent global locomotion. In earlier versions of the system a hormone is propagated through the self-reconfigurable system to achieve synchronization [11]. In later work the hormone is also propagated backwards making all modules synchronized before a new action is initiated [12, 10]. This synchronization takes time $O(n)$ where n is the number of modules. This slows down the system considerably, because it has to be done before each action. Also, the entire system stops working if one hormone is lost. This a significant problem, because a hormone can easily be lost due to unreliable communication, a module disconnecting itself before a response can be given, or a module failure. In fact, the system has n-points of failure which is not desirable. The earlier version is better in this sense, but still performance remains low because a synchronization hormone is sent before each action.

In our system all modules repeatedly go through a cyclic sequence of joint angles describing a motion. This sequence could come from a column in a gait control table, but in our implementation the joint angles are calculated using a cyclic function with period T. Every time a module has completed a given fraction d of the period a message is sent through the child connectors. If the signal is received the child module resets its action sequence making it delayed d compared to the parent. This way the actions of the individual module are decoupled from the synchronization mechanism resulting in a faster and more reliable system. Furthermore, there is no need to make changes to the algorithm if the number of modules changes.

3 General Control Algorithm

We assume that the modules are connected to form a tree structure, that a parent connector is specified, and that this connector is the only one that can connect to child connectors of other modules. Furthermore, we assume that the modules can communicate with the modules to which they are connected.

The algorithm is then used by specifying three components. The first component is a cyclic action sequence $A(t)$. This sequence describes the actions that each module is to repeat cycle after cycle. The second, is the period T of this cycle. The third, is a delay d. This delay specify the fraction of a period the children's action sequences are delayed compared to their parents. The skeleton algorithm looks like this:

```
t = 0
while(1) {
  if (t=d) then <send signal to child connectors>
  if <signal received from parent> then t=0
  <perform action A(t)>
  t = (t+1) modulus T
}
```

Ignoring the first two lines of the loop, the module repeatedly goes through a sequence of actions parameterized by the cyclic counter t. This part of the algorithm alone can make a single module repeatedly perform the specified sequence of actions. In order to coordinate the actions of the individual modules to produce the desired global behavior the modules need to be synchronized. Therefore, at step $t = d$ a signal is send through all child connectors. Note that it does not matter if a child module is actually connected or not. If a child receives a signal it knows that the parent is at $t = d$ and therefore sets its own step counter to $t = 0$. This enforces that the child is delayed d compared to its parent.

From the time the modules are connected it takes time proportional to d times the height of the tree for all the modules to synchronize. To avoid problems with uncoordinated modules initially we make sure the modules do not start moving until they receive the first synchronization signal. After the start-up phase the modules stay synchronized using only constant time.

4 Experimental Setup

To evaluate our algorithm we conducted several experiments using the CONRO modules shown in Figure 1. The CONRO modules have been developed at USC/ISI [3, 5]. The modules are roughly shaped as rectangular boxes measuring 10cm x 4.5cm x 4.5cm and weigh 100grams. The modules have a female connector at one end and three male connectors located at the other. Each connector has a infra-red transmitter and receiver used for local communication and sensing. The modules have two controllable degrees of freedom: pitch (up and down) and yaw (side to side). Processing is taken care of by an onboard Basic Stamp 2 processor. The modules have onboard batteries, but these do not supply enough power for the experiments reported here and therefore the modules are powered through cables. Refer to http://www.isi.edu/conro for more details and videos of the experiments reported later.

5 Experiments

In general, it is a problem how to report performance of a specific part of a self-reconfigurable system because there is such a tight coupling between hardware and software. In this work we choose to report the length of our programs as a measure of the complexity of the control algorithm. This metric is used to support our claim that this control system is minimal. We also report the speed of the locomotion patterns, but this should only be considered an example, the reason being that in our system the limiting factors are how robust the modules physically are, how powerful the motors are, and how much power we can pull from the power source. To report a top speed is not meaningful before we run the robot autonomously on batteries.

5.1 Caterpillar Locomotion

We connect eight of our modules in a chain and designate the male opposite the female connector to be the parent connector. We then implement the algorithm described above with the following parameters.

$$\begin{aligned} T &= 180 \\ pitch(t) &= 50° sin(\tfrac{2\pi}{T} t) \\ yaw(t) &= 0 \\ d &= \tfrac{T}{5} \end{aligned} \tag{1}$$

The motor control of our modules makes the motor go to the desired position as fast as possible. This means that way-points have to be specified to avoid jerky motion. The period T can be used to control the number of way-points and therefore the smoothness and speed of the motion. The action sequence is an oscillation around 0° with an amplitude of 50° and the yaw joint is kept straight. Each module is delayed one fifth of a period compared to its parent.

The modules are connected and after they synchronize a sine wave is traveling along the length of the robot. Refer to Figure 1. This produces caterpillar like locomotion at a speed

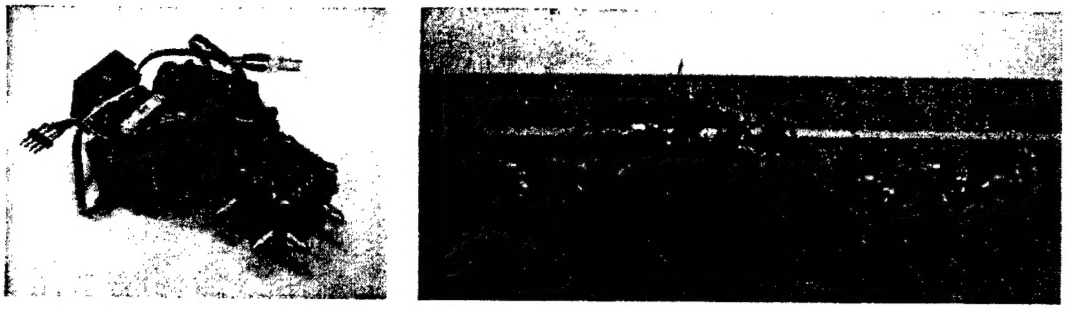

Figure 1: A CONRO module (left). A snapshot of caterpillar like locomotion (right).

of 0.13km/h. Note, that it is easy to adjust the parameters of this motion. For instance, the length of the wave can be controlled using the delay. The program is simple. The main loop contains 16 lines of code excluding comments and labels. The initialization including variable and constant declaration amounts to 18 lines of code.

5.2 Sidewinder Locomotion

We now turn our attention to a locomotion pattern similar to that of a sidewinding snake. A detailed mathematical analysis of this motion pattern has been reported in [2]. Here we just use the intuition that by having modules moving to one side lifted and those moving to the other touching the ground a sidewinder like motion is achieved. The result can be seen in Figure 2. The sidewinder moves at 0.24km/h. The main loop and the initialization contain respectively 19 and 17 lines of code. The parameters used are:

$$\begin{aligned} T &= 180 \\ pitch(t) &= 20°cos(\tfrac{2\pi}{T}t) \\ yaw(t) &= 50°sin(\tfrac{2\pi}{T}t) \\ d &= \tfrac{T}{5} \end{aligned} \tag{2}$$

5.3 Rolling Track Locomotion

If we maintain that each module can only have one parent, but remove the assumption that the structure forms a tree we include loops as structures that can be handled. The rolling track is an example of such a configuration. However, this poses a problem to our algorithm. In the previous experiments we have exploited the assumption that the modules form a tree to implicitly find a conductor. The conductor being the root of the configuration tree. This is a simple mechanism that guarantees that there is one and only one conductor. In a loop configuration this is not the case.

One solution to this problem is to introduce IDs. In our implementation we just make the modules pick a random number and use that as ID. It is not guaranteed to find a unique conductor, but it is a simple solution that works in most cases. The short comings of this approach can easily be avoided if each module has a unique serial number.

The synchronization part of the algorithm now works as before, but it is combined with a simple well-known distributed leader election algorithm [4]. The signals from parent to child now contains a number which is the ID of the module originally sending the signal. Upon receiving a signal a module compares the signal's number to its ID. If it is higher the

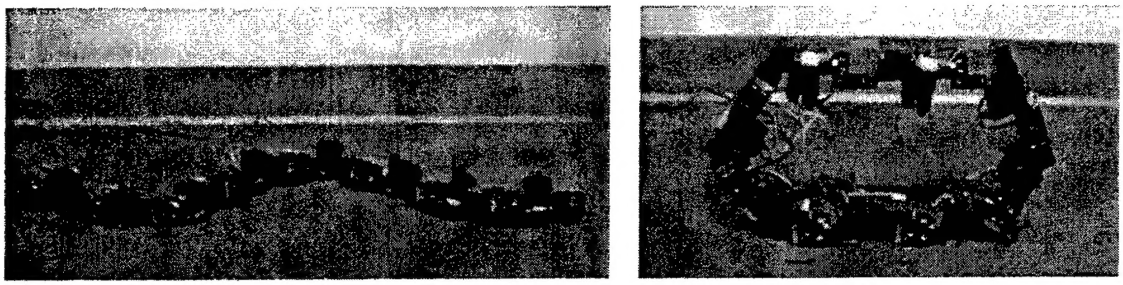

Figure 2: A snapshot of sidewinder like locomotion (left) and the rolling track (right).

module is synchronized and the signal and its ID is propagated along with the synchronization signal. Otherwise, the module consider itself the conductor and ignores the signal. After the system has settled the module with the highest ID dictates the rhythm of the locomotion pattern. The leader election algorithm runs continuously which means that the system quickly synchronizes if modules are replaced. The advantage of combining the algorithms is that there is no need to detect if the conductor fails.

We used this algorithm to implemented the rolling track which can be seen in Figure 2. The rolling track is the fastest gait and achieves a speed of 0.50km/h. The program is now a little more complex and the main loop and initialization contain respectively 35 and 28 lines of code. The parameters for the eight module rolling track is:

$$\begin{aligned} T &= 180 \\ pitch(t) &= \begin{cases} 60°(1 - sin(\frac{2\pi}{T}t)) & if\ t =< \frac{T}{2} \\ 60° & if\ t > \frac{T}{2} \end{cases} \\ yaw(t) &= 0 \\ d &= \frac{T}{4} \end{aligned} \tag{3}$$

Unlike the sidewinder and the caterpillar this control algorithm only works with 8 modules, because of the physical constraint. It might be possible to make a more general solution by making $pitch(t)$ and d a function of the number of modules. The number of modules in the loop could be obtained by the conductor by including a hop count in the signal.

6 Handling a General Configuration

We saw in the previous section that we had to introduce IDs to find a unique conductor in a configuration that contains loops. Introducing the ID mechanism unfortunately ruins the opportunity to use the synchronization algorithm to automatically find a conductor in a tree structure. In fact, the loop algorithm will fail in this situation unless the module with the highest ID also happens to be the root. In order to make a general algorithm the synchronization signal has to be propagated both upwards and downwards in the tree.

7 Discussion

We have presented a general control algorithm and presented examples of how it can be used to achieve three different locomotion patterns. We will now discus some of the properties of this control algorithm.

An important issue in the design of control algorithms for self-reconfigurable robots is if the algorithm scales with the number of modules. The presented algorithm is only initially

dependent on the number of modules, because it decides how long time it takes for the synchronization signal to be propagated through the system. After this start-up phase the time it takes to keep the modules synchronized is independent of the number of modules implying that the algorithm scales. Furthermore, all modules run identical programs making it easier to manage the development process when programming systems consisting of many modules.

The modules of the robot are only loosely coupled through the synchronization signal and therefore the system is highly robust to changes in the number of modules. In fact, the caterpillar can be divided in two and both parts still work. If they are reconnected in a different order they will quickly synchronize to behave as one long caterpillar again. This also implies that the system is robust to module failure. If a module is defect and it can be detected this module can be ejected from the system and the remaining modules when reconnected can continue to perform. Finally, if a synchronization signal is lost it is not crucial for the survival of the system. If a signal is lost it just means that the receiving module and its children will be synchronized a period later.

In the algorithm the synchronization signal is only sent once per period. This means that in order for the modules to stay synchronized the time to complete a period has to be the same for all modules.In the experiments presented here the cycles take the same amount of time, but in more complex control systems where other parts of the control system use random amounts of computation time this can not be assumed to be true. This problem can easily be handle by using timers. Even though timers are not precise enough to keep modules synchronized over a long period of time they can be used for this purpose.

8 Future Work

Our future work will go along two lines. Can the algorithm handle more complex locomotion patterns? We suspect it can be achieved by using different delays through different connectors. For instance, in a multi-legged robot where the head is the conductor, the synchronization signal could travel along the spine modules. When a spine module receives a signal it can first propagate it to the left leg and then the right before propagating the signal to the next spine module.

Another issue is that if the self-reconfigurable robot is to locomote automatically in a real complex environment the control algorithm has to be able to take feedback from the environment into account. A first step in this direction could be to mount sensors on the side of the caterpillar robot and use these to control the yaw joint of the modules.

9 Conclusion

We have presented a general control algorithm for self-reconfigurable robots. The algorithm has the following properties: distributed, scalable, homogeneous, and minimal. We have shown how the algorithm easily can be used to implement a caterpillar and a sidewinder like locomotion pattern. Furthermore, we have seen that with the introduction of IDs in the modules it is possible to handle loop configurations. We have demonstrated this using the rolling track as an example. Finally, we have pointed out interesting lines for future research.

10 Acknowledgements

This work is supported under the DARPA contract DAAN02-98-C-4032, the EU contract IST-20001-33060, and the Danish Technical Research Council contract 26-01-0088.

References

[1] H. Bojinov, A. Casal, and T. Hogg. Emergent structures in modular self-reconfigurable robots. In *Proceedings of the IEEE int. conf. on Robotics & Automation*, volume 2, pages 1734 –1741, San Francisco, USA, 2000.

[2] J.W. Burdick, J. Radford, and G.S. Chirikjian. A 'sidewinding' locomotion gait for hyper-redundant robots. In *Proceedings of the IEEE int. conf. on Robotics & Automation*, pages 101–106, Atlanta, USA, 1993.

[3] A. Castano, R. Chokkalingam, and P. Will. Autonomous and self-sufficient conro modules for reconfigurable robots. In *Proceedings of the 5th int. Symposion on Distributed Autonomous Robotic Systems*, pages 155–164, Knoxville, Texas, USA, 2000.

[4] E. Chang and R. Roberts. An improved algorithm for decentralized extrema-finding in circular configurations of processes. *Communications of the ACM*, 22(5):281–283, 1979.

[5] B. Khoshnevis, B. Kovac, W.-M. Shen, and P. Will. Reconnectable joints for self-reconfigurable robots. In *Proceedings of the IEEE/RSJ int. conf. on Intelligent Robots and Systems*, Maui, Hawaii, USA, 2001.

[6] K. Kotay, D. Rus, M. Vona, and C. McGray. The self-reconfiguring robotic molecule. In *Proceedings of the IEEE int. conf. on Robotics & Automation*, pages 424–431, Leuven, Belgium, 1998.

[7] S. Murata, E. Yoshida, H. Kurokawa, K. Tomita, and S. Kokaji. Self-repairing mechanical systems. *Autonomous Robots*, 10(1):7–21, 2001.

[8] A. Pamecha, C. Chiang, D. Stein, and G.S. Chirikjian. Design and implementation of metamorphic robots. In *Proceedings of the ASME Design Engineering Technical conf. and Computers in Engineering conf.*, pages 1–10, Irvine, USA, 1996.

[9] D. Rus and M. Vona. A physical implementation of the crystalline robot. In *Proceedings of the IEEE int. conf. on Robotics & Automation*, pages 1726–1733, San Francisco, USA, 2000.

[10] B. Salemi, W. Shen, and P. Will. Hormone controlled metamorphic robots. In *Proceedings of the IEEE int. conf. on Robotics & Automation*, pages 4194–4199, Seoul, Korea, 2001.

[11] W.-M. Shen, B. Salemi, and P. Will. Hormone-based control for self-reconfigurable robots. In *Proceedings of the int. conf. on Autonomous Agents*, pages 1–8, Barcelona, Spain, 2000.

[12] W.-M. Shen, B. Salemi, and P. Will. Hormones for self-reconfigurable robots. In *Proceedings of the int. conf. on Intelligent Autonomous Systems*, pages 918–925, Venice, Italy, 2000.

[13] C. Ünsal and P.K. Khosla. Mechatronic design of a modular self-reconfiguring robotic system. In *Proceedings of the IEEE int. conf. on Robotics & Automation*, pages 1742–1747, San Francisco, USA, 2000.

[14] M. Yim. *Locomotion with a unit-modular reconfigurable robot*. PhD thesis, Department of Mechanical Engineering, Stanford University, 1994.

[15] M. Yim, D.G. Duff, and K.D. Roufas. Polybot: A modular reconfigurable robot. In *Proceedings of the IEEE int. conf. on Robotics & Automation*, pages 514–520, San Francisco, USA, 2000.